WIS-CS-236-75
COMPUTER SCIENCES DEPARTMENT
The University of Wisconsin
1210 West Dayton Street
Madison, Wisconsin 53706

Received: January 1975

A WHOLISTIC INTEGRATED COGNITIVE SYSTEM (SEER-T1) THAT INTERACTS WITH ITS ENVIRONMENT OVER TIME

by

Leonard Uhr

Technical Report #236

January 1975

A WHOLISTIC INTEGRATED COGNITIVE SYSTEM (SEER-T1) THAT INTERACTS WITH ITS ENVIRONMENT OVER TIME

Leonard Uhr
University of Wisconsin
Madison, Wisconsin

Abstract

This paper describes, presents and discusses a computer-programmed model for a wholistic system that performs the variety of functions usually thought of as "cognitive" while interacting with an environment of objects that move about and change over time. A relatively sophisticated "recognition cone" system that perceives and describes scenes of moving and changing objects (Uhr, 1975) is embedded in a larger "SEER" (for SEmantic learnER, or Sensed Environment Encoder Recognizer and Responder) system that also associates into memory to respond to queries and access relevant information, deduces solutions to simple problems, parses and "understands" language utterances, effects actions upon the external environment, and learns its memory of transforms and their interconnections.

These functions are kept as simple as possible, and combined in as integrated a way as possible. Each calls upon and helps the others as needed, by implying things, processes and acts into a small number of common lists to which all have access. A unified "cognitive memory" is built up using a single general type of transform, one that is powerful enough to do pattern recognition "feature extraction," scene description "configurational characterizations," memory "associations," deductive "productions," and language "rewrite rules."

Embedding this system in time allows for new interesting and natural appearing interactions among the various processes as they work in parallel and serially call upon one another over time.

The program is coded in EASEy (Uhr, 1973f) an English-like variant of SNOBOL designed to enhance list-processing abilities, and to be easy to read, so that the program itself can be examined and understood.

Introduction and Background

SEER programs are being developed to explore how a wholistic information processing system can perform the variety of cognitive functions (perceiving, remembering, problem-solving, language handling, acting, and learning) that psychologists traditionally study, and that seem to be the components of human thinking.

Research on the Separate Cognitive Processes, and Attempts to Combine Them

Most research in artificial intelligence and mind/brain modelling has looked at

these functions, but separately, in pattern recognition and scene analysis (e.g. Doyle, 1960, Uhr and Vossler, 1961, Andrews et al., 1968, Waltz, 1972, Riseman and Hanson, 1974; see Duda and Hart, 1973, Uhr, 1973a), information retrieval and question-answering (e.g. Lindsay, 1963, Shapiro and Woodmansee, 1969, Woods, 1968, see Simmons, 1965, 1970, Minsky, 1968), deductive problem-solving (e.g. Newell, Shaw and Simon, 1961, Gelernter, 1963, Samuel, 1959, 1969, Robinson, 1965; see Nilsson, 1971, Newell and Simon, 1972), language processing (e.g. Quillian, 1967, Rumelhart and Norman, 1973, Chomsky, 1965; see Anderson and Bower, 1973, Schank and Colby, 1973), acting (e.g. Ernst, 1962; Greene, 1960; see Arbib, 1972), and learning (e.g. Uhr and Vossler, 1961, Samuel, 1959, 1969, Uhr, 1964, Siklossy, 1971, Jordan, 1971, Quillian, 1969, Waterman, 1970, Winston, 1970, Williams, 1974, see Nilsson, 1965, Uhr, 1973a).

There have been a few attempts to combine these separate systems, usually with a focus on acting, as in the robot research (e.g. Nilsson, 1971, Feldman et al., 1971, Winston, 1972, Ejiri et al., 1971; see Ernst, 1970, and Uhr and Kochen, 1969, for critiques). But most of these have built very large and complex systems for each of the major processes of 1) perception of a static scene (input through a television camera), 2) "understanding" a verbal command (input through a teletype), 3) deductive problem-solving, and 4) binding and effecting actions - with very little interaction between these systems. As a result, they have not been able to explore how one process can call upon or help another. For examples, perception should serve to guide an actions-sequence that moves to and grasps an object. But the perceptual system is so rigid that one detailed tv picture is first taken and analyzed, an actions-sequence deduced and then bound to specific movements, and these movements are made blind, without any intermediate feedback.

Steps Toward Integration of Processes, and Generality and Flexibility

Similarly, it would be natural and desirable to have the search for solution paths suggest objects that, if present in the environment, would be useful and in turn focus the attention of the perceptual mechanism on a search for these objects, with perception in turn making use of deductions and memory searches to help guide its search. To do this kind of thing we need simple well-integrated systems. A few small beginnings have been made (e.g. Toda, 1962; Doran, 1968; Uhr and Kochen, 1969). The present system is the latest in a series (Uhr, 1974a,b) that attempts to make a direct attack on this issue - what I will call the problem of everyday thinking about ill-formed problems.

The desire is to achieve systems that are as general, flexible and adaptive as possible. Rather than striving for specific (usually ad hoc) power at a particular task like chess, theorem-proving, or character recognition, SEERs are first-step attempts to develop theory/models of general cognitive systems that can do the variety

of things that people do, in roughly the same integrated way. By keeping SEERs as simple as possible we make them accessible to observation, and to revision. Once a SEER cycles through a variety of simple cognitive tasks, with good interaction and integration among processes, we can begin to strengthen each, or several, processes, by plugging in new functions for them, and testing them separately, and also as part of the thus-augmented system.

Ill-Formed Everyday Thinking Vs. Path-Searching to Solve Well-Formed Problems

Most artificial intelligence research has tried to straightjacket all problems into the paradigm (see Nilsson, 1971) of a search for a path between given(s) (e.g. the initial board in a game, the axioms of a logistic system, the sentence to be parsed) and goal(s) (e.g. the win-states of the game, the theorem to be proved, the "Sentence" node to be reached by the application of some sequence of rewrite rules). The robot projects have similarly tried to decompose the whole cognitive process into separate sub-processes where given(s) and goal(s) are set up, so that they can then be attacked in this way. A good deal of analysis and thought has gone into devising specific heuristics or algorithms for particular search spaces (e.g. perceiving cubes, wedges and pyramids; winning at chess, checkers, or kahlah; answering questions about the weather, troop dispositions, or parts of the body), and these have led to path-searchers that can find fairly deep paths in rather large spaces. Thus problems are made well-formed, and specific techniques are constructed to attack them.

In sharp contrast, most of human thinking has a very different quality. Only rarely do we find somebody making deep deductive searches between well-defined givens and goals. The mathematician's most creative work comes when he finds and posits theorems that appear worthy of proof. It is only because in school we survey a logistic system by studying proofs that we come to think of this as the stuff of mathematics.

Most everyday thinking is much more shallow and, in important ways, ill-formed. Consider typical situations, like figuring out where to go and what to do on a 2-week vacation, weekend, or evening; how to get there; where to eat; what; how to eat it; and so on.

We perceive and interact appropriately with extremely complex mixed fields of words and objects (e.g. the foods, packages, descriptions and prices in a supermarket). Out of this chaos our interests lead us to notice what's relevant, and to (usually shallow) associations and deductions (e.g. "this poundcake mix is easier to make," "that brand's spongecake was soggy," "this mix needs egg yolks, so get eggs and use the whites for pudding"). We move around the store in what might justly be called an ongoing "conversation" where perceived objects and verbiage, memories and deductions all play a role in the constant assessment of relevance that guides our acts and thoughts.

The SEER-T1 System Described

SEER-T1 is the first attempt at an integrated cognitive system that copes with environments of things (words and objects) that move about and change in Time. It builds upon, combines, and extends several previous programs:

SEER systems that respond to static environments have been presented in Uhr, 1974a, 1974b. These extend "DECIDER" systems (Uhr, 1973b, 1973d) that handle environments of mixed objects and words, since in the real world there can be no separate input channels for words and symbols; rather, one of a cognitive system's key problems is to recognize the referential import of certain of the things it perceives.

The perceptual capability has been examined in a sequence of "recognition cone" programs for recognizing and naming (Uhr, 1972) and for describing (Uhr 1973e). A short-term-memory algorithm has been developed to handle successive inputs over time (Uhr, 1973c), and incorporated into the dispersed memory of the recognition cone (Uhr, 1975).

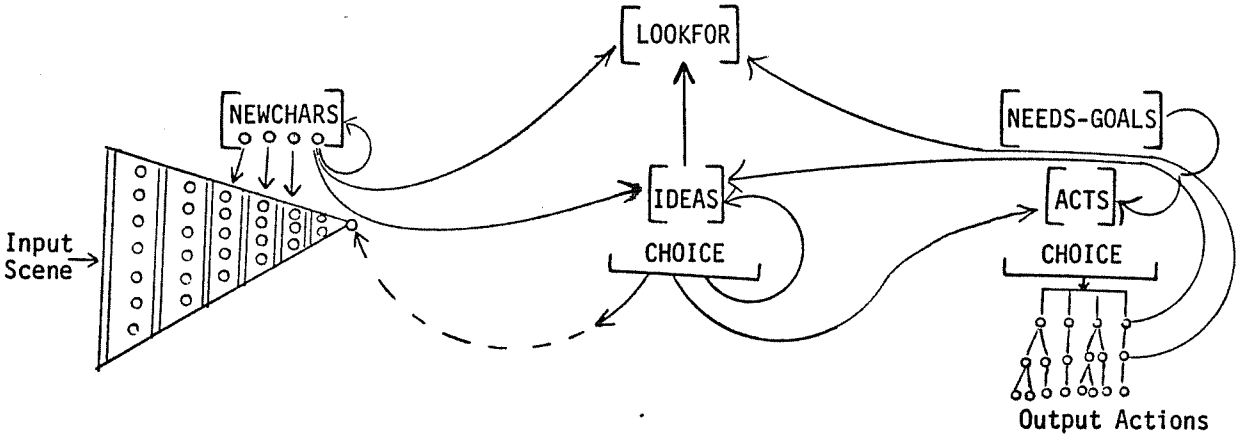
SEER-T1 is the first SEER system that interacts with environments of things that move about and change over Time. Its perceptual abilities are relatively sophisticated, since it incorporates the parallel-serial probabilistic hierarchical "recognition cone" system that handles successive 2-dimensional scenes over time. Time now becomes a major issue for the other cognitive processes as well. The system begins to coordinate the time needed for each process's transformations with the external time in which perceived objects move and change.

This leads to interesting effects and surprising simplifications, since the system can take advantage of continuing time. For example, it is able to mingle inner-directed (top-down) and environment-directed (bottom-up) processes in an especially simple way, by having transforms imply subsequent transforms to apply at any level, but at later moments of time (since it takes time to effect a transform and to send and merge its implications into its output buffer).

Overall Architecture of SEER-T1

A succession of scenes - 2-dimensional arrays, much like the frames of a movie - are input to SEER's Retinal input Buffer, and successively transformed back through the converging layers of the "recognition cone," until all implied things (including external and internal names of wholes, parts, and qualities) have been passed back to the cone's apex (see Uhr, 1972, 1973e for fuller descriptions). Each layer of transforms takes one moment of time. Therefore a 6-layer cone will take 6 moments to transform from Retina to Apex, a 10-layer cone 10 moments, an N-layer cone N moments.

Figure 1. Overall Architecture of a SEER System



At each moment each of the Layers is effecting its Transforms on its input Buffer, and merging its implications into the next output Layer. These Buffers contain things that have been merged into them at previous moments in time, and thus form a set of short-term-memories. They are gradually faded away, the weight of each thing being reduced slightly at each moment, and the thing discarded when its weight goes below a threshold. Each new thing merged into a Buffer cell is given a relatively high initial weight, and this weight is later increased if the same thing is again implied into that cell, by some other transforms and/or at some subsequent moment.

Transforms also imply and merge things into several additional lists - a) a list of things to LOOKFOR, b) a list of dynamically implied transforms (called NEWCHARacterizerS) to apply next (at the next moments of time, until faded away and discarded), and c) a list of IDEAS, which are Transforms to be applied to the central Apex, and serve to associate and deduce new nodes, in a search out into the cognitive memory, d) a list of ACTS that might be effected, and e) a list of NEEDS and GOALS.

At each moment each of these lists is effecting its transforms, in parallel with each other and with each of the Layers in the cone. NEEDS-GOALS implies ACTS and things to LOOKFOR that might satisfy them. Things in LOOKFOR imply NEWCHARacterizerS to apply that might imply them. NEWCHARS are applied at the relative locations specified, and their implications merged into the next Layer of the cone.

The single most highly weighted transform on IDEAS is applied to the Apex and, if it succeeds, its implied things merged back into the Apex, and its implied acts merged into ACTS and implied things-to-lookfor and characterizers-to-apply merged into LOOKFOR and NOWCHARS. And the single most highly weighted act is chosen from ACTS. If it is the same as the previously effected act it continues with the actions-sequence it is presently executing. If it is different, the system generates and binds a new

actions-sequence that it will effect over the next moments of time. This process also implies new IDEAS and things to LOOKFOR to effect these bindings.

Each of these separate processes thus is effected in parallel, at each moment in time, but looking at different buffers and merging their implications into different buffers. The central processes are highly serial, since only the single most highly weighted IDEA and ACT are applied at each moment in time. In sharp contrast, the transforms on all other lists are applied in parallel. We can think of this as a narrow window giving serial central processes, hopefully allowing for more direction as their search is heuristically guided by the combined weights of implications; in contrast to a wide parallel window for the perceptual processes.

Actually, a parameter can be set so that 2, 3, or N most highly weighted IDEAs can be applied in parallel, at the same moment. We will need to test for an optimum window size, one that uses parallel processes to speed up search but without losing too much of the directedness given by serial decisions.

Flow of Processes over Time

This quite naturally gives interactions among all the processes. But these interactions take the time needed to effect the required sequence of transforms. For example, a characterizing transform might imply a top-angle, and also that the system LOOKFOR a T and for the word THE. The symbol T implies that features of T be applied, and these transforms will be effected 2 moments in time later. But they can be effected at any Layer of the cone. If at the next more central Layer, it will be on the transformed scene only 1 moment later (plus the short-term memory of still-not-deleted earlier moments), if at a Layer 2 more peripheral on the transformed scene 4 moments later (but closer to the raw input, thus giving a top-down direction to processing).

Similarly, THE implies H and E a little to the right of the T, which in turn imply their features, etc. At the same time, other things that have been merged into LOOKFOR (because implied by previous transforms, NEEDS, GOALS, and ACTs being effected) imply transforms to be applied.

Thus there is no need to explicitly decide to "look up" then "look down" with feedback loops moving to lower layers. Rather, the general capability to imply things into any lists, plus the continuing application of implied transforms as time advances, gives the possibility of any mixture of top-down and bottom-up and $list_a$-to-$list_b$ processes.

Microstructure of the General Configurational Transform

All transforms have the same general structure. A transform consists of a set of conditions (things to look for with optional weights, relative locations and other attributes, and tests to effect), a threshold for success, and a set of implications

(things, transforms, and/or triggers; with optional weights, relative locations and other attributes). The weights and thresholds give the probabilistic quality that seems to give the power and robustness to real-world pattern recognizers. But when the threshold is set to equal the combined weights of all the conditions the transform becomes deterministic, insisting upon an "anding" of all parts, and can therefore be used for the typically deterministic syntactic rewrite rules, semantic transformations, deductive productions, and memory associations.

Weights seem desirable for association and deduction as well for perception, and this general configurational transform allows us to explore probabilistic searches. The weights serve to give heuristic guidance to the search, since the system chooses to effect the most highly weighted transform on IDEAS, and that weight will be a function of any contextually related transforms, from no matter what source, that implied that transform.

Rewrite rules and productions, which use the simple relations of concatenation (left-right connectedness) and order can also be expressed with these transforms, by using relative locations. But a wider variety of what I think are potentially very useful "perceptual rewrite rules" are possible, rules that specify less rigid relations among the parts - the sort of relations we need for real-world language, where noise and ungrammatical utterances are common.

Demonstration Examples of SEER-T1's Behavior

Uhr, 1974a, 1974b gives examples of SEER's behavior, including recognizing, describing, answering, deducing, finding and manipulating, in response to single static scenes of objects and verbal utterances mixed together. SEER-T1 will handle all of these even if the objects and/or the words appear only parts at a time, in successive scenes. For example, utterances might come in a word, or a letter or two, or a phoneme or two, at each moment. An object like a dog might come in snout, face, ears, body, legs, then tail, as though the perceptual field were smaller than the object. SEER-T1 works so long as the utterance or object comes completely into view before its first parts and their transforms have been faded and discarded from short-term-memories. This means that the parameters of the short-term-memory algorithm must be adjusted to handle the expected speed of arrival and disappearance of external things. That is, internal processes must be tuned to the same time range as the external processes they attempt to handle.

Thus SEER-T1, when given the transforms shown in the Appendix, handles the following simple example problems, along with many other problems. (When given more transforms, for more words, objects and associations, it will handle more. The expectation is that these transforms will be learned, rather than pre-programmed. Specific transforms have the same status as do the specific rewrite rules for a specific language's

vocabulary and grammar, that are given to a parser.)

The following is an example of a simple movement:

T_1	TOUCH			
T_2	THE	@APPLE	@PAIL	
T_3	BANANA	@APPLE	@PAIL	
T_4		@APPLE	@PAIL	@BANANA
T_5		@APPLE	@PAIL	@BANANA
.				
.				
.				
Output Time:		@APPLE	@PAIL	:@BANANA:

Note that @ indicates an object, as opposed to a word - any arbitrary picture or symbol could be used; colons (:...:) are used to indicate a thing is touched.

The following examplifies a simple manipulation:

T_1	PUT	@APPLE	@PAIL	@BOX	@BANANA
T_2	BANANA IN	@APPLE	@PAIL	@BOX	@BANANA
T_3	PAIL	@APPLE	@PAIL	@BOX	@BANANA
.					
.					
.					
Output Time:		@APPLE	(@BANANA	@PAIL)	@BOX

Note that the things manipulated need <u>not</u> be present until after the command is completed.

An <u>internal</u> need can similarly lead to an action (e.g. HUNGER-NEED implies LOOKFOR FOOD, implies LOOKFOR BANANA, implies LOOKFOR @BANANA implies the movement of touching, as does an external command.

Discussion

Implicit vs. Explicit Indicants of Time

Note that no explicit indication is stored about the time of first merging of a thing into a cell. It would be quite simple to have a clock advance at each moment, and to store that clock's time with each newly implied thing, and use relative time as part of the conditions for success of a transform. This would make the discrimination of things like D;O;G from G;O;D quite simple and straightforward. But it seems preferable to explore whether the present simpler mechanism may not be sufficient, since the relative weights of the parts will be a function of time.

The Size of the Perceptual and Cognitive Windows

Exploration is needed into the optimal balance between large parallel windows and deep serial processes. In addition to widening their windows, to allow for more parallel processes in the central IDEAS and ACTS lists, we must examine the effects of limiting the number of parallel transforms at each layer. This is ultimately necessary when we think of the actual physical parallel-serial computer (e.g., a nerve network) being modelled. Only a limited volume of physical space is available for connections. And there is a tradeoff between having many transforms in a single layer, and having more serial layers, each with fewer transforms.

Some Possible Mechanisms for Focussing Attention

It seems likely that associative memory searches should be more parallel than deductive searches for paths. This suggests the possibility of handling them with two separate IDEAS lists, and/or dynamically changing the size of the window as a function of characteristics of the problem being tackled.

Alternately, a "line of thought" indicated by a high weight could establish a new list that temporarily replaces IDEAS, and lets the system in a sense concentrate on a single much narrowed search.

Summary

This paper describes a wholistic cognitive system that begins to explore the ways in which the usually separated processes of perception, associative remembering, deductive problem-solving, language handling and learning can be integrated to handle the relatively ill-formed kinds of problems encountered in everyday thinking. SEER-T1 interacts with environments that move and change over time. It can perceive moving and changing environments of words and objects mixed together, and respond to verbal utterances that are input over time. Its associative and deductive thinking, and its actions similarly take time to effect.

Some simple examples of the system's behavior are given, to show how it handles the different cognitive processes as a function of sequences of input scenes within which it must recognize a mixture of words and objects, and also of its own internal needs, goals and expectations.

A variety of different configurations are now open to exploration, since the interaction among the different processes can be varied in a number of ways. Each process will also be varied and, hopefully, strengthened, and examined in the test-bed of the whole system. Learning mechanisms will be added, so that as much as possible of the system's set of transforms, and its overall architecture, can be discovered through experience in interacting with its environment. Its layered hierarchical parallel-serial architecture and its single general type of probabilistic configurational

transform have been designed with flexibility, generality, and learning in mind.

References

Anderson, J. R. and Bower, G. H. Human Associative Memory, Washington: Winston, 1973.

Arbib, M. A. The Metaphorical Brain, New York: Wiley, 1972.

Chomsky, N. Aspects of the Theory of Syntax, Cambridge: MIT Press, 1965.

Doran, J. E. Experiments with a pleasure-seeking automaton, Machine Intelligence 3, Edinburgh: Univ. of Edinburgh Press, 1968.

Ejiri, M., Uno, T., Yoda, H., Goto, T. and Takeyasu, K. An intelligent robot with cognition and decision-making ability, Proc. 2d Joint Int. Conf. on Artificial Intell., 1971, London, 350-358.

Ernst, H. A. MH-1, a computer-operated hand, Proc. SJCC, 1962, 20, 39-45.

Ernst, H. A. Computer-controlled robots, IBM Report RC 2781, Yorktown Heights, 1970.

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

Gelernter, H. Realization of a geometry-theorem proving machine, In E. Feigenbaum and J. Feldman, Eds., Computers and Thought, New York: McGraw-Hill, 1963, 134-152.

Greene, P. H. A suggested model for information representation in a computer that perceives, learns and reasons, Proc. FJCC, 1960, 17, 151-164.

Jordan, S. R. Learning to Use Contextual Patterns in Language Processing. Unpubl. Ph.D. Diss., Univ. of Wisconsin, 1971.

Lindsay, R. K. Inferential memory as the basis of machines which understand natural language, In E. Feigenbaum and J. Feldman, Eds., Computers and Thought, New York: McGraw-Hill, 1963, 217-233.

Minsky, M. (Ed.) Semantic Information Processing, Cambridge: MIT Press, 1968.

Newell, A., Shaw, C. and Simon, H. A. GPS, a program that simulates human thought, In: Lennende Automaten, Munich: R. Oldenbourg, 1961. (In Feigenbaum and Feldman.)

Newell, A. and Simon, H. A. Human Problem Solving. Englewood Cliffs, N. J.: Prentice-Hall, 1972.

Nilsson, N. J. Learning Machines. New York: McGraw-Hill, 1965.

Nilsson, N. J. Problem-Solving Methods in Artificial Intelligence, New York: McGraw-Hill, 1971.

Quillian, M. R. Word concepts: a theory and simulation of some basic semantic capabilities, Behav. Sci., 1967, 12, 410-430.

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

Riseman, E. M. and Hanson, A. R. Design of a semantically directed vision processor, COINS Tech Rept. 74-C1, Univ. of Mass., 1974.

Robinson, J. A. A machine-oriented logic based on the resolution principle, J.ACM, 1965, 12, 23-41.

Rumelhart, D. E. and Norman, D. A. Active semantic networks as a model of human memory, Proc. 3d Int. Joint Conf. on Artificial Intell., Palo Alto, 1973, 450-457.

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

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

Schank, R. and Colby, K. M. (Eds.) Computer Models of Thought and Language, San Francisco: Freeman, 1973.

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

Siklossy, L. A language-learning heuristic program. Cognitive Psychology, 1971, 2, 479-495.

Simmons, R. Answering English questions by computer: a survey, Comm. ACM, 1965, 8, 53-70.

Simmons, R. Natural language question-answering systems: 1969, Comm. ACM, 1970, 13, 15-30.

Toda, M. The design of a fungus eater, Behav. Sci., 1962, 7, 164-183.

Uhr, L. Pattern-string learning programs. Behavioral Science, 1964, 9, 258-270.

Uhr, L. "Recognition cones" that perceive and describe scenes that move and change over time. Computer Sci. Dept. Tech. Rept., Univ. of Wisconsin, 1975.

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

Waterman, D. A. Generalization techniques for automating the learning of heuristics. Artificial Intelligence, 1970, 1, 121-170.

Williams, H. A Hierarchical Net-Structure Learning System for Pattern Description, Unpubl. Ph.D. Diss., Univ. of Wisconsin, Madison, 1974.

Winston, P. H. Learning Structural Descriptions from Examples. Unpubl. Ph.D. Diss., MIT, 1970.

Woods, W. A. Procedural semantics for a question-answering machine. Proc. Fall Joint Computer Conf., 1968, 33, 457-471.

Acknowledgements

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.

Appendix: The EASEy Program for SEER-T1

```
( SEER-T1, for scenes that change over time.
        NLAYERS = 7                                                    M1
        FADER = 'F O O R C F '                                         M2
        C = 'CHARSNEXT'                                                M3
        G = 'GOALS'                                                    M4
        I = 'IDEASNEXT'                                                M5
        A = 'ACTSNEXT'                                                 M6
        N = 'NEEDS' [IN]                                               M7
(Moves from central apex to the peripheral retina
NEXT-TIME  MERGE( NEED-CHANGES FLAILS )                                 1
        MERGE( POINTAT( ABOVE( NEEDS GOALS , 5 ) EXPECTATIONS LOOKFOR ) )  2
        MERGE( CHARSNEXT , 'NEWCHARS' )                                 3
        MERGE( IDEASNEXT , 'IDEAS' )                                    4
        MERGE( ACTSNEXT , 'ACTS' )                                      5
        erase  CHARSNEXT, IDEASNEXT, ACTSNEXT, STEPS                    6
        FADE( 'IDEAS ACTS GOALS CHARS NEWCHARS LOOKFOR EXPECTATIONS ' )  7
( CHOOSE and  execute most highly weighted act
ACT     CHOOSE( ACTS )                                                  8
( Test whether this choice has a higher TOTAL weight than the ACTIVE act.
        is TOTAL lessthan ACTIVEWT ?  [+ ACTON]                         9
        ACTIVEWT = TOTAL + 5                                            10
        is THING sameas ACTIVE ? [+ ACTON]                              11
        ACTIVE = THING                                                  12
        erase ACTIONS                                                   13
      from  $THING get TYPE THRESH '%D=' DESCR 'I=' IMPLIEDS %
( Does 3 steps in the chosen act at each moment.
ACTON   ACTTRIES = 3
ACT     is 0 lessthan ACTTRIES ? yes- ACTTRIES = ACTTRIES - 1  [ - THINK ]
```

```
          from IMPLIEDS get ACTION ARGS ; = [- OUT]
ACT2      from HIST get CLASS ARG = [ + $ACTION ]
            ARG = ARGS  [ $ACTION]
(OUTputs its actions-sequence
OUT       is TOOUT sameas EMPTY ? [+ SEARCH ]
          output          TOOUT
          output          EXTERNAL [ IN]
(Initiates a SEARCH to help in completing the frustrated act.
SEARCH    from $ARG get '%I=' IMPLIEDS % [- FAIL ]
SEARCH2   from IMPLIEDS get IMPLIED WT ; = [- FAIL]
          from IMPLIED get NEEDED '$' [-SEARCH2 ]
          from $NEEDED get '%D=" PARTS %
          MERGE(POINTAT(PARTS),N)

RETURNACT  on NEWCHARS set CHOOSE [ TMORE ]
FAIL       output 'FAILED' EXTERNAL  [IN ]
D          [T]
(Names the most highly implied object of class specified in ARGument
T          from $(L;0;0) get CHOOSE($(L;0;0),ARG) = [ -ACT ]
(TOTAL weight must be above 5.
           is TOTAL greaterthan 5 ? [-ACT]
           on TOOUT list THING        [$('AC' ACTION) ]
ACD        on TOOUT set '( WITH '
(Describes the scene
D3         from HISTC get CLASS THINGH =  [-D2 ]
           from  $(L;0;0) get : that THINGH REST = [-D3]
           on TOOUT list THINGH ; [D3]
D2         on TOOUT set ' )  '  [T ]
(Finds the first THING for each ARGument.
F          from  $(L;0;0) get # that ARG # REST : THING MORE ] = [ -ACT2 ]
(Finds THING only if without variations in EXTERNAL input
           from EXTERNAL get that THING = : THING : [-ACT2 ]
           on TOOUT list  'A ' THING ' IS FOUND- '  [ACT2]
(Moves all Found things as PREPosition (TO or FROM) indicates
M          is CLASS sameas  'PREP' ? [ -F ]
           from HIST get CLASS ARGT     [-ACT ]
           from  $(L;0;0) get # that ARGT # REST : TARGET  [-SEARCH ]
M2         from EXTERNAL get : THINGA : = [+ $('M' ARG) ]
MTO        from EXTERNAL get that TARGET = '(' TARGET THINGA ')' [M2 ]
MFROM      from EXTERNAL get LEFT that TARGET RIGHT =
+             : 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]
             MAXWT = TOTAL
R2         on TOOUT list THING ;
           from $(L;0;0) get CHOOSE($(L;0;0),ARG) = [ -ACT ]
           is TOTAL lessthan MAXWT / 2 ? [+ACT - R2 ]
SEARCHR    MERGE($(L;0;0),IDEAS) [ RETURNACT ]
(Computes
C          from ARGS get OP CLASSA CLASSB
           from $(L;0;0) get # that OP # REST : OP
           from $(L;0;0) get # that CLASSA  # REST : ARGA [ - SEARCHC ]
           from $(L;0;0) get # that CLASSB # REST : ARGB [ + $OP ]
SEARCHC    MERGE(POINTAT(INTEGERS) [RETURNACT ]
(ADD,- etc. are OPS.
ADD        on TOOUT list ARGA + ARGB  [ACT]
-          on TOOUT list ARGA - ARGB  [ACT]
/          on TOOUT list ARGA / ARGB  [ACT]
```

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


( Applies the N most highly weighted Transforms on IDEAS to the apex.
THINK   N = 5
THINKAGAIN  is N lessthan 1 ? [+ PERCEIVE ]
                N = N - 1
        from IDEAS get CLASSES : TRANS TOTAL HIST ] = [ + PERCEIVE ]
        erase LOC
        from $TRANS get THRESH '%D=' DESCR '%I=' IMPLIEDS %
TH1     from DESCR get  : CLASS WT DC ] = [ - EVAL ]
        integer( DC ) [ - TH3 ]
        NEAREST( CLASS , LOC + DC , $( 0 ; 0 ; 0 ) ) [ +TH4  - TH1 ]
TH3     NEAREST(  CLASS , 'Z' , $( 0 . 0 . 0 ) ) [ - TH1 ]
TH4     TOTAL = TOTAL + ( WT * WTL )  / DIST
        on HIST list  CLASS THING [ TH1 ]
EVAL    is TOTAL lessthan  THRESH ?  [ + THINKAGAIN ]
        MERGE( IMPLIEDS , I , HIST , TOTAL / THRESH )  [ THINK ]
( Moves through the recognition cone, from apex ($(0 ; 0 ; 0 ) ) to retina
PERCEIVE  L = 1
          R = 1
          C = 1
TREPEAT  TODO = CHARS LAYERS
T6      from TODO get RSTEP CSTEP NOWDO % =  [ - TREPEAT ]
        erase COUNT, INTENSITY
        at start of NOWDO set ':FADER X ]:NORMALIZE ' RSTEP * CSTEP ] NEWCHARS
T5      from NOWDO get CLASSES : TRANS TOTAL HIST ] = [ - ITER ]
        at start of  TRANS get  : DRT : DCT : = [ + T8 ]
        erase DRT DCT
```

```
T8      from $TRANS get  RA CAA RMAX CMAX DO
        RA = RA + DRT
T7      CA = CAA + DCT
T4      from $DO get TYPE THRESH '%D=' DESCR '%I=' IMPLIEDS %
        erase  GOT  TOTAL
        X = L - 1 ; ( RA / RSTEP ) ; ( CA / CSTEP )

T3      from DESCR get : CLASS TVAL DR DC ] = [ - $(TYPE 2 ) ]
        BEFORE =  L ; ( RA + DR ) ; ( CA + DC )
( Checks whether BEFORE cell is empty, e.g. outside the cone
        is $BEFORE sameas  EMPTY ?  [ + T3 - $(TYPE 2 ) ]
( TYPE A Transforms that average and difference
A1      MERGE( $BEFORE , $X , , TVAL , CLASS )  [ T3 ]
( TYPE T Transforms that look for configurations
T1      from $BEFORE  get # that CLASS # REST : THING VAL =  [ - T3 ]
        is TVAL lessthan VAL ? yes-  TOTAL = TOTAL + VAL - TVAL
        on GOT list  CLASS THING  [ T3 ]
T2      is TOTAL lessthan  THRESH ?  [ + A2 ]
        MERGE( IMPLIEDS , $X , GOT )  [ A2 ]
( TYPE D transforms, that trigger decisions directly
D1      MERGE( CHOOSE(  $BEFORE ) , $( 0 ; 0 ; 0 ) )  [ A2 ]
A2      is CA lessthan  $CMAX + DCT ? yes- CA = CA + 1  [ + T4 ]
        is RA lessthan  $RMAX + DRT ? yes- RA = RA + 1  [ + T7  - T% ]
(FADEs the earlier layer, only one fade per cell
F1      FADE( BEFORE )  [  A2 ]
ITER    is L lessthan NLAYERS ?  yes- L = L + 1  [ - NEXT ]
        R = R * RSTEP
        C = C * CSTEP
        on STEPS list : L RSTEP CSTEP  [ T6 ]
(Input the next moment of time
NEXT    erase RA
IN      input TYPE DESCR ]  [ + $('M' TYPE ) - end ]
```

```
MSENSE   erase CA
         on EXTERNAL list DESCR
S1       from DESCR get and call SPOTSIZE symbols  SPOT = [ - S2 ]
         integer(SPOT)  [ - S4 ]
         MERGE( 'BRIGHT QUAL :BRIGHT ' SPOT ] , $( L ; RA ; CA ) ) [ S3 ]
( Stores colors or any other primitives
S4       from SPOT get PRIM VAL =  [ - S6 ]
         from $PRIM get '%I=' IMPLIEDS '%C=' CLASSES  %  [ - S5 ]
         MERGE( IMPLIEDS PRIM CLASSES : PRIM VAL ] , $( L ; RA ; CA ) ) [ S4 ]
S5       MERGE( PRIM : PRIM VAL ] , $( L ; RA ; CA ) )  [ S4 ]

( To handle primitives that are letter-strings, e.g. sentences
S6       is SPOT sameas EMPTY ?  [ + S3 ]
         MERGE( SPOT 'SYMB :SYMB ' SPOT 5 ] , $( L ; RA ; CA ) )  [ S3 ]
S3       CA = CA + 1  [ S1 ]
S2       is RA lessthan R ?  yes-  RA = RA + 1  [ + IN ]
         output 'ONE MOMENT HAS BEEN INPUT.'  [ NEXT-TIME ]

( The following functions are used by the main program.
(FADEs many lists gradually.  FADE will contain all THINGs discarded.
FADE     DEFINE: FADE( LISTS , FADE )
FA1      from LISTS get  LISTA  =  [ - return ]
         TOFADE = $LISTA
         erase $LISTA
FA2      from TOFADE get LEFT : THING WT RIGHT ]  =  [ - FA1 ]
FA4      is WT lessthan 1 ?  [ - FA3 ]
         on FADE list LEFT : THING WT RIGHT ]  [ FA2 ]
( Will divide by 2 if FADE factor is not specified.
FA3      on $LISTA list LEFT : THING WT / ( FADE + 2 ) RIGHT ]  [ FA2 ]
```

```
(MERGE two lists, combining weights and HISTories
MERGE       (DEFINE: MERGE(LISTA,LISTB,HISTA,WT,CLASS)
            is CLASS sameas EMPTY ? [-ME1]
            from LISTA get LEFT : THING WTA HIST ] = [ - return + ME3 ]
ME1         from LISTA get LEFT # that CLASS # REST : THING WTA HIST] = [ -return]
ME3         from WTA get '$' = TOTAL
(Can optionally specify LISTB as part of THING
            from THING get '$' LISTB =
            is LISTB sameas 'CH' ? [ + CH ]
            from $LISTB get : that THING TOTAL HISTB =
+             NAME TOTAL + (WT+1).* WTL HISTA HISTB ;   [+ME1 ]
            from $THING get '%C=' CLASSES %  [+ME2 ]
              erase CLASSES
ME2         on $LISTB list THING CLASSES : THING (WT+1) * WTL LISTB HISTA ] [ME1]
CH          at start of CENTRAL list CHOOSE($(L;RA;CA),THING)  [ME1 ]
(Back-links only to transforms with names
POINTAT     DEFINE: POINTAT(THINGS)
PA1         from THINGS get CLASSES : THING HIST ]  = [- return ]
            from $THING get '%D=' TOTHINGS % [- PA1]
            MERGE(TOTHINGS;N)        [ PA1 ]
(NORMALIZE to keep weights roughly constant even though converging passed-on things

ABS         DEFINE: ABS(ABS)
ABS1        at start of ABS get '-' = [return]
(CHOOSE MAX or MIN weighted (MAX if no type is specified)
CHOOSE      DEFINE: CHOOSE(LISTA,CLASS,TYPE)
(CLASS can be a specific thing, or empty (in which case all things are chosen
(among)
CH1         from LISTA get CLASSES : FIRST THWT HIHIST ] = [ - -return ]
            is CLASS sameas EMPTY ? [+ CH2]
            from CLASSES get # that CLASS # [-CH1 ]
            list CHOOSE = CLASSES: FIRST THWT L ; RA ; CA HIHIST
CH2         from LISTA get ORCLASSES : ORTHING ORWT ORHIST ] = [+CH4]
            from CHOOSE get CLASSES : THING TOTAL LOC HIST ] [return ]
CH4         is CLASS sameas EMPTY ? [+ $('CH'TYPE)
            from ORCLASSES get # that CLASS #  [+$('CH' TYPE)  -CH2 ]
CHMIN       is ORWT lessthan THWT ? yes - THWT = ORWT [+CH3 - CH2]
CH          [CHMAX ]
CHMAX       is THWT lessthan ORWT ? yes- THWT = ORWT  [+CH3 - CH2 ]
CH2         list CHOOSE = ORCLASSES : ORTHING ORWT LISTA HIHIST ] [CH2]
```

A Note on EASEy Programs (See Uhr, 1973f 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 on 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 [-return] goto.

A WHOLISTIC INTEGRATED COGNITIVE SYSTEM (SEER-T1) THAT INTERACTS WITH ITS ENVIRONMENT OVER TIME

Leonard Uhr
University of Wisconsin
Madison, Wisconsin

Abstract

This paper describes, presents and discusses a computer-programmed model for a wholistic system that performs the variety of functions usually thought of as "cognitive" while interacting with an environment of objects that move about and change over time. A relatively sophisticated "recognition cone" system that perceives and describes scenes of moving and changing objects (Uhr, 1975) is embedded in a larger "SEER" (for SEmantic learnER, or Sensed Environment Encoder Recognizer and Responder) system that also associates into memory to respond to queries and access relevant information, deduces solutions to simple problems, parses and "understands" language utterances, effects actions upon the external environment, and learns its memory of transforms and their interconnections.

These functions are kept as simple as possible, and combined in as integrated a way as possible. Each calls upon and helps the others as needed, by implying things, processes and acts into a small number of common lists to which all have access. A unified "cognitive memory" is built up using a single general type of transform, one that is powerful enough to do pattern recognition "feature extraction," scene description "configurational characterizations," memory "associations," deductive "productions," and language "rewrite rules."

Embedding this system in time allows for new interesting and natural appearing interactions among the various processes as they work in parallel and serially call upon one another over time.

The program is coded in EASEy (Uhr, 1973f) an English-like variant of SNOBOL designed to enhance list-processing abilities, and to be easy to read, so that the program itself can be examined and understood.

Introduction and Background

SEER programs are being developed to explore how a wholistic information processing system can perform the variety of cognitive functions (perceiving, remembering, problem-solving, language handling, acting, and learning) that psychologists traditionally study, and that seem to be the components of human thinking.

Research on the Separate Cognitive Processes, and Attempts to Combine Them

Most research in artificial intelligence and mind/brain modelling has looked at

these functions, but separately, in pattern recognition and scene analysis (e.g. Doyle, 1960, Uhr and Vossler, 1961, Andrews et al., 1968, Waltz, 1972, Riseman and Hanson, 1974; see Duda and Hart, 1973, Uhr, 1973a), information retrieval and question-answering (e.g. Lindsay, 1963, Shapiro and Woodmansee, 1969, Woods, 1968, see Simmons, 1965, 1970, Minsky, 1968), deductive problem-solving (e.g. Newell, Shaw and Simon, 1961, Gelernter, 1963, Samuel, 1959, 1969, Robinson, 1965; see Nilsson, 1971, Newell and Simon, 1972), language processing (e.g. Quillian, 1967, Rumelhart and Norman, 1973, Chomsky, 1965; see Anderson and Bower, 1973, Schank and Colby, 1973), acting (e.g. Ernst, 1962; Greene, 1960; see Arbib, 1972), and learning (e.g. Uhr and Vossler, 1961, Samuel, 1959, 1969, Uhr, 1964, Siklossy, 1971, Jordan, 1971, Quillian, 1969, Waterman, 1970, Winston, 1970, Williams, 1974, see Nilsson, 1965, Uhr, 1973a).

There have been a few attempts to combine these separate systems, usually with a focus on acting, as in the robot research (e.g. Nilsson, 1971, Feldman et al., 1971, Winston, 1972, Ejiri et al., 1971; see Ernst, 1970, and Uhr and Kochen, 1969, for critiques). But most of these have built very large and complex systems for each of the major processes of 1) perception of a static scene (input through a television camera), 2) "understanding" a verbal command (input through a teletype), 3) deductive problem-solving, and 4) binding and effecting actions - with very little interaction between these systems. As a result, they have not been able to explore how one process can call upon or help another. For examples, perception should serve to guide an actions-sequence that moves to and grasps an object. But the perceptual system is so rigid that one detailed tv picture is first taken and analyzed, an actions-sequence deduced and then bound to specific movements, and these movements are made blind, without any intermediate feedback.

Steps Toward Integration of Processes, and Generality and Flexibility

Similarly, it would be natural and desirable to have the search for solution paths suggest objects that, if present in the environment, would be useful and in turn focus the attention of the perceptual mechanism on a search for these objects, with perception in turn making use of deductions and memory searches to help guide its search. To do this kind of thing we need simple well-integrated systems. A few small beginnings have been made (e.g. Toda, 1962; Doran, 1968; Uhr and Kochen, 1969). The present system is the latest in a series (Uhr, 1974a,b) that attempts to make a direct attack on this issue - what I will call the problem of everyday thinking about ill-formed problems.

The desire is to achieve systems that are as general, flexible and adaptive as possible. Rather than striving for specific (usually ad hoc) power at a particular task like chess, theorem-proving, or character recognition, SEERs are first-step attempts to develop theory/models of general cognitive systems that can do the variety

of things that people do, in roughly the same integrated way. By keeping SEERs as simple as possible we make them accessible to observation, and to revision. Once a SEER cycles through a variety of simple cognitive tasks, with good interaction and integration among processes, we can begin to strengthen each, or several, processes, by plugging in new functions for them, and testing them separately, and also as part of the thus-augmented system.

Ill-Formed Everyday Thinking Vs. Path-Searching to Solve Well-Formed Problems

Most artificial intelligence research has tried to straightjacket all problems into the paradigm (see Nilsson, 1971) of a search for a path between given(s) (e.g. the initial board in a game, the axioms of a logistic system, the sentence to be parsed) and goal(s) (e.g. the win-states of the game, the theorem to be proved, the "Sentence" node to be reached by the application of some sequence of rewrite rules). The robot projects have similarly tried to decompose the whole cognitive process into separate sub-processes where given(s) and goal(s) are set up, so that they can then be attacked in this way. A good deal of analysis and thought has gone into devising specific heuristics or algorithms for particular search spaces (e.g. perceiving cubes, wedges and pyramids; winning at chess, checkers, or kahlah; answering questions about the weather, troop dispositions, or parts of the body), and these have led to path-searchers that can find fairly deep paths in rather large spaces. Thus problems are made well-formed, and specific techniques are constructed to attack them.

In sharp contrast, most of human thinking has a very different quality. Only rarely do we find somebody making deep deductive searches between well-defined givens and goals. The mathematician's most creative work comes when he finds and posits theorems that appear worthy of proof. It is only because in school we survey a logistic system by studying proofs that we come to think of this as the stuff of mathematics.

Most everyday thinking is much more shallow and, in important ways, ill-formed. Consider typical situations, like figuring out where to go and what to do on a 2-week vacation, weekend, or evening; how to get there; where to eat; what; how to eat it; and so on.

We perceive and interact appropriately with extremely complex mixed fields of words and objects (e.g. the foods, packages, descriptions and prices in a supermarket). Out of this chaos our interests lead us to notice what's relevant, and to (usually shallow) associations and deductions (e.g. "this poundcake mix is easier to make," "that brand's spongecake was soggy," "this mix needs egg yolks, so get eggs and use the whites for pudding"). We move around the store in what might justly be called an ongoing "conversation" where perceived objects and verbiage, memories and deductions all play a role in the constant assessment of relevance that guides our acts and thoughts.

The SEER-T1 System Described

SEER-T1 is the first attempt at an integrated cognitive system that copes with environments of things (words and objects) that move about and change in Time. It builds upon, combines, and extends several previous programs:

SEER systems that respond to static environments have been presented in Uhr, 1974a, 1974b. These extend "DECIDER" systems (Uhr, 1973b, 1973d) that handle environments of mixed objects and words, since in the real world there can be no separate input channels for words and symbols; rather, one of a cognitive system's key problems is to recognize the referential import of certain of the things it perceives.

The perceptual capability has been examined in a sequence of "recognition cone" programs for recognizing and naming (Uhr, 1972) and for describing (Uhr 1973e). A short-term-memory algorithm has been developed to handle successive inputs over time (Uhr, 1973c), and incorporated into the dispersed memory of the recognition cone (Uhr, 1975).

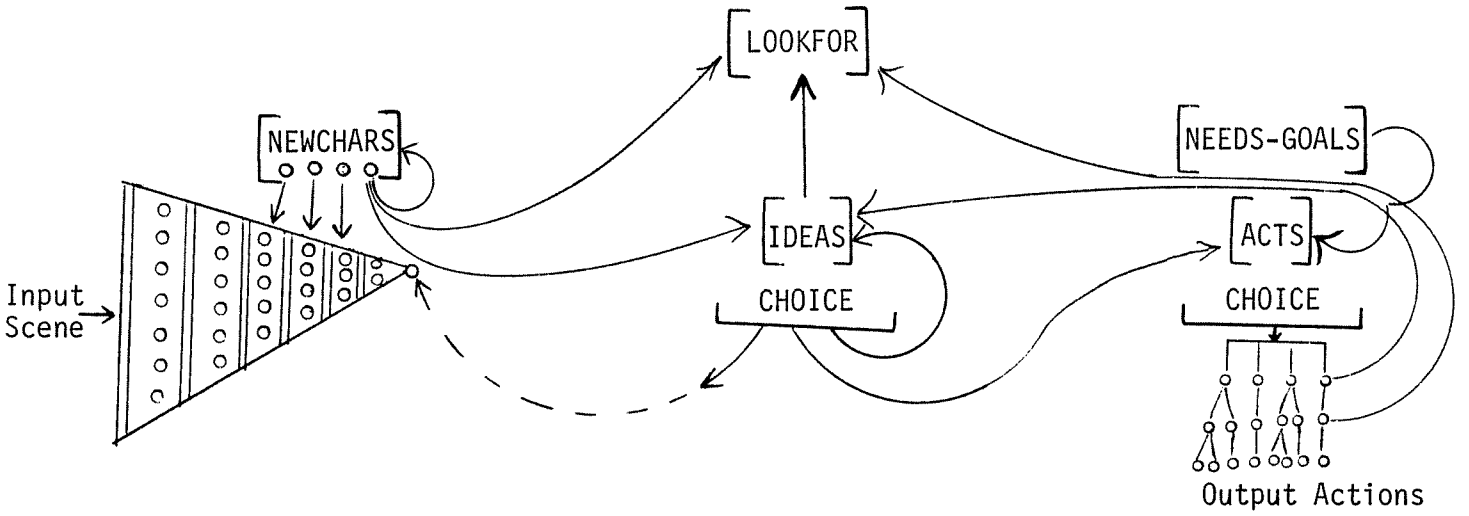
SEER-T1 is the first SEER system that interacts with environments of things that move about and change over Time. Its perceptual abilities are relatively sophisticated, since it incorporates the parallel-serial probabilistic hierarchical "recognition cone" system that handles successive 2-dimensional scenes over time. Time now becomes a major issue for the other cognitive processes as well. The system begins to coordinate the time needed for each process's transformations with the external time in which perceived objects move and change.

This leads to interesting effects and surprising simplifications, since the system can take advantage of continuing time. For example, it is able to mingle inner-directed (top-down) and environment-directed (bottom-up) processes in an especially simple way, by having transforms imply subsequent transforms to apply at any level, but at later moments of time (since it takes time to effect a transform and to send and merge its implications into its output buffer).

Overall Architecture of SEER-T1

A succession of scenes - 2-dimensional arrays, much like the frames of a movie - are input to SEER's Retinal input Buffer, and successively transformed back through the converging layers of the "recognition cone," until all implied things (including external and internal names of wholes, parts, and qualities) have been passed back to the cone's apex (see Uhr, 1972, 1973e for fuller descriptions). Each layer of transforms takes one moment of time. Therefore a 6-layer cone will take 6 moments to transform from Retina to Apex, a 10-layer cone 10 moments, an N-layer cone N moments.

Figure 1. Overall Architecture of a SEER System



At each moment each of the Layers is effecting its Transforms on its input Buffer, and merging its implications into the next output Layer. These Buffers contain things that have been merged into them at previous moments in time, and thus form a set of short-term-memories. They are gradually faded away, the weight of each thing being reduced slightly at each moment, and the thing discarded when its weight goes below a threshold. Each new thing merged into a Buffer cell is given a relatively high initial weight, and this weight is later increased if the same thing is again implied into that cell, by some other transforms and/or at some subsequent moment.

Transforms also imply and merge things into several additional lists - a) a list of things to LOOKFOR, b) a list of dynamically implied transforms (called NEWCHARacterizerS) to apply next (at the next moments of time, until faded away and discarded), and c) a list of IDEAS, which are Transforms to be applied to the central Apex, and serve to associate and deduce new nodes, in a search out into the cognitive memory, d) a list of ACTS that might be effected, and e) a list of NEEDS and GOALS.

At each moment each of these lists is effecting its transforms, in parallel with each other and with each of the Layers in the cone. NEEDS-GOALS implies ACTS and things to LOOKFOR that might satisfy them. Things in LOOKFOR imply NEWCHARacterizerS to apply that might imply them. NEWCHARS are applied at the relative locations specified, and their implications merged into the next Layer of the cone.

The single most highly weighted transform on IDEAS is applied to the Apex and, if it succeeds, its implied things merged back into the Apex, and its implied acts merged into ACTS and implied things-to-lookfor and characterizers-to-apply merged into LOOKFOR and NOWCHARS. And the single most highly weighted act is chosen from ACTS. If it is the same as the previously effected act it continues with the actions-sequence it is presently executing. If it is different, the system generates and binds a new

actions-sequence that it will effect over the next moments of time. This process also implies new IDEAS and things to LOOKFOR to effect these bindings.

Each of these separate processes thus is effected in parallel, at each moment in time, but looking at different buffers and merging their implications into different buffers. The central processes are highly serial, since only the single most highly weighted IDEA and ACT are applied at each moment in time. In sharp contrast, the transforms on all other lists are applied in parallel. We can think of this as a narrow window giving serial central processes, hopefully allowing for more direction as their search is heuristically guided by the combined weights of implications; in contrast to a wide parallel window for the perceptual processes.

Actually, a parameter can be set so that 2, 3, or N most highly weighted IDEAs can be applied in parallel, at the same moment. We will need to test for an optimum window size, one that uses parallel processes to speed up search but without losing too much of the directedness given by serial decisions.

Flow of Processes over Time

This quite naturally gives interactions among all the processes. But these interactions take the time needed to effect the required sequence of transforms. For example, a characterizing transform might imply a top-angle, and also that the system LOOKFOR a T and for the word THE. The symbol T implies that features of T be applied, and these transforms will be effected 2 moments in time later. But they can be effected at any Layer of the cone. If at the next more central Layer, it will be on the transformed scene only 1 moment later (plus the short-term memory of still-not-deleted earlier moments), if at a Layer 2 more peripheral on the transformed scene 4 moments later (but closer to the raw input, thus giving a top-down direction to processing).

Similarly, THE implies H and E a little to the right of the T, which in turn imply their features, etc. At the same time, other things that have been merged into LOOKFOR (because implied by previous transforms, NEEDS, GOALS, and ACTs being effected) imply transforms to be applied.

Thus there is no need to explicitly decide to "look up" then "look down" with feedback loops moving to lower layers. Rather, the general capability to imply things into any lists, plus the continuing application of implied transforms as time advances, gives the possibility of any mixture of top-down and bottom-up and $list_a$-to-$list_b$ processes.

Microstructure of the General Configurational Transform

All transforms have the same general structure. A transform consists of a set of conditions (things to look for with optional weights, relative locations and other attributes, and tests to effect), a threshold for success, and a set of implications

(things, transforms, and/or triggers; with optional weights, relative locations and other attributes). The weights and thresholds give the probabilistic quality that seems to give the power and robustness to real-world pattern recognizers. But when the threshold is set to equal the combined weights of all the conditions the transform becomes deterministic, insisting upon an "anding" of all parts, and can therefore be used for the typically deterministic syntactic rewrite rules, semantic transformations, deductive productions, and memory associations.

Weights seem desirable for association and deduction as well for perception, and this general configurational transform allows us to explore probabilistic searches. The weights serve to give heuristic guidance to the search, since the system chooses to effect the most highly weighted transform on IDEAS, and that weight will be a function of any contextually related transforms, from no matter what source, that implied that transform.

Rewrite rules and productions, which use the simple relations of concatenation (left-right connectedness) and order can also be expressed with these transforms, by using relative locations. But a wider variety of what I think are potentially very useful "perceptual rewrite rules" are possible, rules that specify less rigid relations among the parts - the sort of relations we need for real-world language, where noise and ungrammatical utterances are common.

Demonstration Examples of SEER-T1's Behavior

Uhr, 1974a, 1974b gives examples of SEER's behavior, including recognizing, describing, answering, deducing, finding and manipulating, in response to single static scenes of objects and verbal utterances mixed together. SEER-T1 will handle all of these even if the objects and/or the words appear only parts at a time, in successive scenes. For example, utterances might come in a word, or a letter or two, or a phoneme or two, at each moment. An object like a dog might come in snout, face, ears, body, legs, then tail, as though the perceptual field were smaller than the object. SEER-T1 works so long as the utterance or object comes completely into view before its first parts and their transforms have been faded and discarded from short-term-memories. This means that the parameters of the short-term-memory algorithm must be adjusted to handle the expected speed of arrival and disappearance of external things. That is, internal processes must be tuned to the same time range as the external processes they attempt to handle.

Thus SEER-T1, when given the transforms shown in the Appendix, handles the following simple example problems, along with many other problems. (When given more transforms, for more words, objects and associations, it will handle more. The expectation is that these transforms will be learned, rather than pre-programmed. Specific transforms have the same status as do the specific rewrite rules for a specific language's

vocabulary and grammar, that are given to a parser.)

The following is an example of a simple movement:

T_1	TOUCH			
T_2	THE	@APPLE	@PAIL	
T_3	BANANA	@APPLE	@PAIL	
T_4		@APPLE	@PAIL	@BANANA
T_5		@APPLE	@PAIL	@BANANA
.				
.				
.				
Output Time:		@APPLE	@PAIL	:@BANANA:

Note that @ indicates an object, as opposed to a word - any arbitrary picture or symbol could be used; colons (:...:) are used to indicate a thing is touched.

The following examplifies a simple manipulation:

T_1	PUT	@APPLE	@PAIL	@BOX	@BANANA
T_2	BANANA IN	@APPLE	@PAIL	@BOX	@BANANA
T_3	PAIL	@APPLE	@PAIL	@BOX	@BANANA
.					
.					
.					
Output Time:		@APPLE	(@BANANA	@PAIL)	@BOX

Note that the things manipulated need <u>not</u> be present until after the command is completed.

An <u>internal</u> need can similarly lead to an action (e.g. HUNGER-NEED implies LOOKFOR FOOD, implies LOOKFOR BANANA, implies LOOKFOR @BANANA implies the movement of touching, as does an external command.

Discussion

Implicit vs. Explicit Indicants of Time

Note that no explicit indication is stored about the time of first merging of a thing into a cell. It would be quite simple to have a clock advance at each moment, and to store that clock's time with each newly implied thing, and use relative time as part of the conditions for success of a transform. This would make the discrimination of things like D;O;G from G;O;D quite simple and straightforward. But it seems preferable to explore whether the present simpler mechanism may not be sufficient, since the relative weights of the parts will be a function of time.

The Size of the Perceptual and Cognitive Windows

Exploration is needed into the optimal balance between large parallel windows and deep serial processes. In addition to widening their windows, to allow for more parallel processes in the central IDEAS and ACTS lists, we must examine the effects of limiting the number of parallel transforms at each layer. This is ultimately necessary when we think of the actual physical parallel-serial computer (e.g., a nerve network) being modelled. Only a limited volume of physical space is available for connections. And there is a tradeoff between having many transforms in a single layer, and having more serial layers, each with fewer transforms.

Some Possible Mechanisms for Focussing Attention

It seems likely that associative memory searches should be more parallel than deductive searches for paths. This suggests the possibility of handling them with two separate IDEAS lists, and/or dynamically changing the size of the window as a function of characteristics of the problem being tackled.

Alternately, a "line of thought" indicated by a high weight could establish a new list that temporarily replaces IDEAS, and lets the system in a sense concentrate on a single much narrowed search.

Summary

This paper describes a wholistic cognitive system that begins to explore the ways in which the usually separated processes of perception, associative remembering, deductive problem-solving, language handling and learning can be integrated to handle the relatively ill-formed kinds of problems encountered in everyday thinking. SEER-T1 interacts with environments that move and change over time. It can perceive moving and changing environments of words and objects mixed together, and respond to verbal utterances that are input over time. Its associative and deductive thinking, and its actions similarly take time to effect.

Some simple examples of the system's behavior are given, to show how it handles the different cognitive processes as a function of sequences of input scenes within which it must recognize a mixture of words and objects, and also of its own internal needs, goals and expectations.

A variety of different configurations are now open to exploration, since the interaction among the different processes can be varied in a number of ways. Each process will also be varied and, hopefully, strengthened, and examined in the test-bed of the whole system. Learning mechanisms will be added, so that as much as possible of the system's set of transforms, and its overall architecture, can be discovered through experience in interacting with its environment. Its layered hierarchical parallel-serial architecture and its single general type of probabilistic configurational

transform have been designed with flexibility, generality, and learning in mind.

References

Anderson, J. R. and Bower, G. H. Human Associative Memory, Washington: Winston, 1973.

Arbib, M. A. The Metaphorical Brain, New York: Wiley, 1972.

Chomsky, N. Aspects of the Theory of Syntax, Cambridge: MIT Press, 1965.

Doran, J. E. Experiments with a pleasure-seeking automaton, Machine Intelligence 3, Edinburgh: Univ. of Edinburgh Press, 1968.

Ejiri, M., Uno, T., Yoda, H., Goto, T. and Takeyasu, K. An intelligent robot with cognition and decision-making ability, Proc. 2d Joint Int. Conf. on Artificial Intell., 1971, London, 350-358.

Ernst, H. A. MH-1, a computer-operated hand, Proc. SJCC, 1962, 20, 39-45.

Ernst, H. A. Computer-controlled robots, IBM Report RC 2781, Yorktown Heights, 1970.

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

Gelernter, H. Realization of a geometry-theorem proving machine, In E. Feigenbaum and J. Feldman, Eds., Computers and Thought, New York: McGraw-Hill, 1963, 134-152.

Greene, P. H. A suggested model for information representation in a computer that perceives, learns and reasons, Proc. FJCC, 1960, 17, 151-164.

Jordan, S. R. Learning to Use Contextual Patterns in Language Processing. Unpubl. Ph.D. Diss., Univ. of Wisconsin, 1971.

Lindsay, R. K. Inferential memory as the basis of machines which understand natural language, In E. Feigenbaum and J. Feldman, Eds., Computers and Thought, New York: McGraw-Hill, 1963, 217-233.

Minsky, M. (Ed.) Semantic Information Processing, Cambridge: MIT Press, 1968.

Newell, A., Shaw, C. and Simon, H. A. GPS, a program that simulates human thought, In: Lennende Automaten, Munich: R. Oldenbourg, 1961. (In Feigenbaum and Feldman.)

Newell, A. and Simon, H. A. Human Problem Solving. Englewood Cliffs, N. J.: Prentice-Hall, 1972.

Nilsson, N. J. Learning Machines. New York: McGraw-Hill, 1965.

Nilsson, N. J. Problem-Solving Methods in Artificial Intelligence, New York: McGraw-Hill, 1971.

Quillian, M. R. Word concepts: a theory and simulation of some basic semantic capabilities, Behav. Sci., 1967, 12, 410-430.

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

Riseman, E. M. and Hanson, A. R. Design of a semantically directed vision processor, COINS Tech Rept. 74-C1, Univ. of Mass., 1974.

Robinson, J. A. A machine-oriented logic based on the resolution principle, J.ACM, 1965, 12, 23-41.

Rumelhart, D. E. and Norman, D. A. Active semantic networks as a model of human memory, Proc. 3d Int. Joint Conf. on Artificial Intell., Palo Alto, 1973, 450-457.

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

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

Schank, R. and Colby, K. M. (Eds.) Computer Models of Thought and Language, San Francisco: Freeman, 1973.

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

Siklossy, L. A language-learning heuristic program. Cognitive Psychology, 1971, 2, 479-495.

Simmons, R. Answering English questions by computer: a survey, Comm. ACM, 1965, 8, 53-70.

Simmons, R. Natural language question-answering systems: 1969, Comm. ACM, 1970, 13, 15-30.

Toda, M. The design of a fungus eater, Behav. Sci., 1962, 7, 164-183.

Uhr, L. Pattern-string learning programs. Behavioral Science, 1964, 9, 258-270.

Uhr, L. "Recognition cones" that perceive and describe scenes that move and change over time. Computer Sci. Dept. Tech. Rept., Univ. of Wisconsin, 1975.

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

Waterman, D. A. Generalization techniques for automating the learning of heuristics. Artificial Intelligence, 1970, 1, 121-170.

Williams, H. A Hierarchical Net-Structure Learning System for Pattern Description, Unpubl. Ph.D. Diss., Univ. of Wisconsin, Madison, 1974.

Winston, P. H. Learning Structural Descriptions from Examples. Unpubl. Ph.D. Diss., MIT, 1970.

Woods, W. A. Procedural semantics for a question-answering machine. Proc. Fall Joint Computer Conf., 1968, 33, 457-471.

Acknowledgements

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.

Appendix: The EASEy Program for SEER-T1

```
( SEER-T1, for scenes that change over time.
        NLAYERS = 7                                                    M1
        FADER = 'F O O R C F '                                         M2
        C = 'CHARSNEXT'                                                M3
        G = 'GOALS'                                                    M4
        I = 'IDEASNEXT'                                                M5
        A = 'ACTSNEXT'                                                 M6
        N = 'NEEDS' [IN]                                               M7
(Moves from central apex to the peripheral retina
NEXT-TIME  MERGE( NEED-CHANGES FLAILS )                                 1
        MERGE( POINTAT( ABOVE( NEEDS GOALS , 5 ) EXPECTATIONS LOOKFOR ) )  2
        MERGE( CHARSNEXT , 'NEWCHARS' )                                 3
        MERGE( IDEASNEXT , 'IDEAS' )                                    4
        MERGE( ACTSNEXT , 'ACTS' )                                      5
        erase  CHARSNEXT, IDEASNEXT, ACTSNEXT, STEPS                    6
        FADE( 'IDEAS ACTS GOALS CHARS NEWCHARS LOOKFOR EXPECTATIONS ' )  7
( CHOOSE and  execute most highly weighted act
ACT     CHOOSE( ACTS )                                                  8
( Test whether this choice has a higher TOTAL weight than the ACTIVE act.
        is  TOTAL lessthan  ACTIVEWT ?   [+ ACTON]                      9
        ACTIVEWT = TOTAL + 5                                            10
        is  THING sameas  ACTIVE ? [+  ACTON]                           11
        ACTIVE = THING                                                  12
        erase  ACTIONS                                                  13
      from  $THING get TYPE THRESH '%D=' DESCR 'I=' IMPLIEDS %
( Does 3 steps in the chosen act at each moment.
ACTON   ACTTRIES = 3
ACT     is 0 lessthan ACTTRIES ? yes- ACTTRIES = ACTTRIES - 1  [  - THINK ]
```

```
          from IMPLIEDS get ACTION ARGS ; = [- OUT]
ACT2      from HIST get CLASS ARG = [ + $ACTION ]
            ARG = ARGS  [ $ACTION]
(OUTputs its actions-sequence
OUT       is TOOUT sameas EMPTY ? [+ SEARCH ]
          output          TOOUT
          output          EXTERNAL [IN]
(Initiates a SEARCH to help in completing the frustrated act.
SEARCH    from $ARG get '%I=' IMPLIEDS % [- FAIL ]
SEARCH2   from IMPLIEDS get IMPLIED WT ; = [- FAIL ]
          from IMPLIED get NEEDED '$' [-SEARCH2 ]
          from $NEEDED get '%D=" PARTS %
          MERGE(POINTAT(PARTS),N)

RETURNACT  on NEWCHARS set CHOOSE [ TMORE ]
FAIL       output 'FAILED' EXTERNAL   [IN ]
D          [T]
(Names the most highly implied object of class specified in ARGument
T          from $(L;0;0) get CHOOSE($(L;0;0),ARG) = [ -ACT ]
(TOTAL weight must be above 5.
           is TOTAL greaterthan 5 ? [-ACT]
           on TOOUT list THING        [$('AC' ACTION) ]
ACD        on TOOUT set '( WITH '
(Describes the scene
D3         from HISTC get CLASS THINGH =   [-D2 ]
           from  $(L;0;0) get : that THINGH REST = [-D3]
           on TOOUT list THINGH ; [D3 ]
D2         on TOOUT set ' )   '  [T ]
(Finds the first THING for each ARGument.
F          from  $(L;0;0) get # that ARG # REST : THING MORE ] = [ -ACT2 ]
(Finds THING only if without variations in EXTERNAL input
           from EXTERNAL get that THING = : THING : [-ACT2 ]
           on TOOUT list  'A ' THING ' IS FOUND- '  [ACT2]
(Moves all Found things as PREPosition (TO or FROM) indicates
M          is CLASS sameas  'PREP' ? [ -F ]
           from HIST get CLASS ARGT     [ -ACT ]
           from  $(L;0;0) get # that ARGT # REST : TARGET  [-SEARCH ]
M2         from EXTERNAL get : THINGA : = [+ $('M' ARG) ]
MTO        from EXTERNAL get that TARGET = '(' TARGET THINGA ')' [M2 ]
MFROM      from EXTERNAL get LEFT that TARGET RIGHT =
+             : 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]
             MAXWT = TOTAL
R2         on TOOUT list THING ;
           from $(L;0;0) get CHOOSE($(L;0;0),ARG) = [ -ACT ]
           is TOTAL lessthan MAXWT / 2 ? [+ACT - R2 ]
SEARCHR    MERGE($(L;0;0),IDEAS) [ RETURNACT ]
(Computes
C          from ARGS get OP CLASSA CLASSB
           from $(L;0;0) get # that OP # REST : OP
           from $(L;0;0) get # that CLASSA  # REST : ARGA [ - SEARCHC ]
           from $(L;0;0) get # that CLASSB # REST : ARGB [ + $OP ]
SEARCHC    MERGE(POINTAT(INTEGERS) [RETURNACT ]
(ADD,-etc. are OPS.
ADD        on TOOUT list ARGA + ARGB  [ACT]
-          on TOOUT list ARGA - ARGB  [ACT]
/          on TOOUT list ARGA / ARGB  [ACT]
```

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


( Applies the N most highly weighted Transforms on IDEAS to the apex.

THINK    N = 5

THINKAGAIN  is N lessthan 1 ? [+ PERCEIVE ]

              N = N - 1

         from IDEAS get CLASSES : TRANS TOTAL HIST ] =   [ + PERCEIVE ]

         erase  LOC

         from $TRANS get THRESH '%D=' DESCR '%I=' IMPLIEDS %

TH1      from DESCR get  : CLASS WT DC ] =  [ - EVAL ]

         integer( DC ) [ - TH3 ]
         NEAREST( CLASS , LOC + DC , $( 0 ; 0 ; 0 ) ) [ +TH4  - TH1 ]
TH3      NEAREST(  CLASS , 'Z' , $( 0 . 0 . 0 ) ) [ - TH1 ]

TH4      TOTAL = TOTAL + ( WT * WTL )  / DIST

         on HIST list  CLASS THING  [ TH1 ]

EVAL     is TOTAL  lessthan  THRESH ?  [ + THINKAGAIN ]

         MERGE( IMPLIEDS , I , HIST , TOTAL / THRESH )  [ THINK ]

( Moves through the recognition cone, from apex ($(0 ; 0 ; 0 ) ) to retina

PERCEIVE  L = 1

          R = 1

          C = 1

TREPEAT   TODO = CHARS LAYERS

T6        from TODO get RSTEP CSTEP NOWDO % =  [ - TREPEAT ]

          erase COUNT, INTENSITY

          at start of NOWDO set ':FADER X ]:NORMALIZE ' RSTEP * CSTEP ] NEWCHARS

 T5        from NOWDO get CLASSES : TRANS TOTAL HIST ] =  [ - ITER ]

           at start of  TRANS get  : DRT : DCT : =  [ + T8 ]

           erase DRT DCT
```

```
T8      from $TRANS get  RA CAA RMAX CMAX DO
        RA = RA + DRT
T7      CA = CAA + DCT
T4      from $DO get TYPE THRESH '%D=' DESCR '%I=' IMPLIEDS %
        erase  GOT  TOTAL
        X = L - 1 ; ( RA / RSTEP ) ; ( CA / CSTEP )

T3      from DESCR get : CLASS TVAL DR DC ] = [ - $(TYPE 2 ) ]
        BEFORE =  L ; ( RA + DR ) ; ( CA + DC )
( Checks whether BEFORE cell is empty, e.g. outside the cone
        is $BEFORE sameas  EMPTY ?  [ + T3 - $(TYPE 2 ) ]
( TYPE A Transforms that average and difference
A1      MERGE( $BEFORE , $X , , TVAL , CLASS )  [ T3 ]
( TYPE T Transforms that look for configurations
T1      from $BEFORE  get # that CLASS # REST : THING VAL =  [ - T3 ]
        is TVAL lessthan VAL ? yes-  TOTAL = TOTAL + VAL - TVAL
        on GOT list  CLASS THING  [ T3 ]
T2      is TOTAL lessthan  THRESH ?  [ + A2 ]
        MERGE( IMPLIEDS , $X , GOT )  [ A2 ]
( TYPE D transforms, that trigger decisions directly
D1      MERGE( CHOOSE(  $BEFORE ) , $( 0 ; 0 ; 0 ) )  [ A2 ]
A2      is CA lessthan  $CMAX + DCT ? yes- CA = CA + 1  [ + T4 ]
        is RA lessthan  $RMAX + DRT ? yes- RA = RA + 1  [ + T7  - T% ]
(FADEs the earlier layer, only one fade per cell
F1      FADE( BEFORE )  [  A2 ]
ITER    is L lessthan NLAYERS ?  yes- L = L + 1  [ - NEXT ]
        R = R * RSTEP
        C = C * CSTEP
        on STEPS list : L RSTEP CSTEP  [ T6 ]
(Input the next moment of time
NEXT    erase RA
IN      input TYPE DESCR ]  [ + $('M' TYPE ) - end ]
```

```
SENSE   erase CA
        on EXTERNAL list DESCR
1       from DESCR get and call SPOTSIZE symbols  SPOT =  [ - S2 ]
        integer(SPOT)  [ - S4 ]
        MERGE( 'BRIGHT QUAL :BRIGHT ' SPOT ] , $( L ; RA ; CA ) ) [ S3 ]
( Stores colors or any other primitives
4       from SPOT get PRIM VAL =  [ - S6 ]
        from $PRIM get '%I=' IMPLIEDS '%C=' CLASSES  %  [ - S5 ]
        MERGE( IMPLIEDS PRIM CLASSES : PRIM VAL ] , $( L ; RA ; CA ) ) [ S4 ]
5       MERGE( PRIM : PRIM VAL ] , $( L ; RA ; CA ) )  [ S4 ]

 To handle primitives that are letter-strings, e.g. sentences

6       is SPOT sameas EMPTY ?  [ + S3 ]
        MERGE( SPOT 'SYMB :SYMB ' SPOT 5 ] , $( L ; RA ; CA ) )  [ S3 ]
3       CA = CA + 1  [ S1 ]
2       is RA lessthan R ?  yes-  RA = RA + 1  [ + IN ]
        output 'ONE MOMENT HAS BEEN INPUT.'  [ NEXT-TIME ]

 The following functions are used by the main program.
FADEs many lists gradually.  FADE will contain all THINGs discarded.
'ADE    DEFINE: FADE( LISTS , FADE )
'A1     from LISTS get  LISTA  =  [ - return ]
        TOFADE = $LISTA
        erase $LISTA
'A2     from TOFADE get LEFT : THING WT RIGHT ]  =  [ - FA1 ]
'A4     is WT lessthan 1 ?  [ - FA3 ]
        on FADE list LEFT : THING WT RIGHT ]  [ FA2 ]
 Will divide by 2 if FADE factor is not specified.
'A3     on $LISTA list LEFT : THING WT / ( FADE + 2 ) RIGHT ]  [ FA2 ]
```

```
(MERGE two lists, combining weights and HISTories
MERGE       (DEFINE: MERGE(LISTA,LISTB,HISTA,WT,CLASS)
            is CLASS sameas EMPTY ? [-ME1]
            from LISTA get LEFT : THING WTA HIST ] = [ - return + ME3 ]
ME1         from LISTA get LEFT # that CLASS # REST : THING WTA HIST] = [ -return]
ME3         from WTA get '$' = TOTAL
(Can optionally specify LISTB as part of THING
            from THING get '$' LISTB =
            is LISTB sameas 'CH' ? [ + CH ]
            from $LISTB get : that THING TOTAL HISTB =
+             NAME TOTAL + (WT+1) * WTL HISTA HISTB ;   [+ME1 ]
            from $THING get '%C=' CLASSES %  [+ME2 ]
              erase CLASSES
ME2         on $LISTB list THING CLASSES : THING (WT+1) * WTL LISTB HISTA ] [ME1]
CH          at start of CENTRAL list CHOOSE($(L;RA;CA),THING)  [ME1 ]
(Back-links only to transforms with names
POINTAT     DEFINE: POINTAT(THINGS)
PA1         from THINGS get CLASSES : THING HIST ]  = [- return ]
            from $THING get '%D=' TOTHINGS % [- PA1]
            MERGE(TOTHINGS;N)        [ PA1 ]
(NORMALIZE to keep weights roughly constant even though converging passed-on things

ABS         DEFINE: ABS(ABS)
ABS1        at start of ABS get '-' = [return]
(CHOOSE MAX or MIN weighted (MAX if no type is specified)
CHOOSE      DEFINE: CHOOSE(LISTA,CLASS,TYPE)
(CLASS can be a specific thing, or empty (in which case all things are chosen
(among)
CH1         from LISTA get CLASSES : FIRST THWT HIHIST ] = [ - -return ]
            is CLASS sameas EMPTY ? [+ CH2]
            from CLASSES get # that CLASS # [-CH1 ]
            list CHOOSE = CLASSES: FIRST THWT L ; RA ; CA HIHIST
CH2         from LISTA get ORCLASSES : ORTHING ORWT ORHIST ] = [+CH4]
            from CHOOSE get CLASSES : THING TOTAL LOC HIST ] [return ]
CH4         is CLASS sameas EMPTY ? [+ $('CH'TYPE)
            from ORCLASSES get # that CLASS #  [+$('CH' TYPE)  -CH2 ]
CHMIN       is ORWT lessthan THWT ? yes - THWT = ORWT [+CH3 - CH2]
CH          [CHMAX ]
CHMAX       is THWT lessthan ORWT ? yes- THWT = ORWT  [+CH3 - CH2 ]
CH2         list CHOOSE = ORCLASSES : ORTHING ORWT LISTA HIHIST ] [CH2]
```

A Note on EASEy Programs (See Uhr, 1973f 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 on 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 [-return] goto.