ARTIFICIAL INTELLIGENCE SYSTEMS THAT UNDERSTAND

Herbert A. Simon
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

From its beginnings, artificial intelligence has borrowed freely from the vocabulary of psychology. The use of the word "intelligence" to label our area of research is a case in point. Other terms referring originally to human mental processes that have considerable currency in AI are "thinking," "comprehending," and, with increasing frequency in the past five years, "understanding." In fact, these terms are probably used more freely in AI than in experimental psychology, where a deep suspicion of "mentalistic" terminology still lingers as a heritage of behaviorism.

It is not my intent to engage in a barren lexicographic exercise, nor to bait those among us who are aroused to indignant emotion whenever terms from human psychology are used in reference to computers. We employ these anthropomorphic terms because we find them useful in defining our research goals, and therefore it is important that we attach clear operational meanings to them. Before discussing computer programs that understand, we need to consider how we can judge whether such programs are successful.

When we attribute intelligence to a computer, we mean that it is doing something which, if done by a human, would be called "intel 1 igent" (Felgenbaun and Feldman, 1963, p.3)- The criteria we use to judge whether a man or woman is acting intelligently are generally external criteria; we do not often attach electrodes to our subject's head, much less open it up to peer inside. Instead, we ask questions like, "Is he behaving appropriately to the circumstances?" or "Is she solving the problem?"[1]

We do not get much help from either experimental psychology or ordinary speech in defining "understanding" in a precise way. It is doubtful that there is a single, unambiguous concept lying behind the diversity of usage. This vagueness of everyday language should prepare us to encounter a multiplicity of different kinds of understanding, and a corresponding multiplicity of criteria that we must apply to behavior (whether of a computer or a human being) to test whether understanding has been achieved.

[1]The discussion here is limited to the goals of AI, as usually defined by its practitioners. On the other hand, when we are using the computer as an instrument of psychological research, to simulate human thinking, the criteria for defining intelligence are stricter. In the latter case we must insist on similarity of process as well as similarity of product. See A. Newell and H. A. Simon, Human Problem Solving, ch. 1-5.

Moore and Newell (1974), p.203) have offered a portmanteau definition of understanding as a relation between a system, S, and some knowledge, K:

> S understands knowledge K if S uses K whenever appropriate.

While the definition is comprehensive, it does not help us decide what is an "appropriate" use of knowledge. Appropriateness must be judged in relation to the tasks the system is required to perform. A system may understand multiplication well enough to find the product of 3 and 4, but not well enough to apply the distributive law to numerical expressions, or to state how multiplication is defined by Peano's axioms for the positive intergers.

In everyday language, we speak not only of understanding knowledge (e.g., "I understood the textbook pretty well."), but also of understanding how to do a task; and so may define:

> S understands task T if S has the knowledge and procedures needed to perform T.

From these two definitions, we see that understanding is a three-termed relation among S, K, and T, and that either the knowledge, K, or the task, T, may be taken as the object of the understanding. We may expect, therefore, that research progress in constructing systems that understand will consist in enriching both the bodies of knowledge available to the systems, and the procedures for using that knowledge in the performance of wider and wider ranges of tasks. Knowledge without appropriate procedure for its use is dumb, and procedure without suitable knowledge is blind.

Some Simple Understanding Capabilities

The plan of this paper will be to illustrate the reciprocal interaction between knowledge and task in systems that understand, with examples of artificial intelligence programs that incorporate varying amounts and kinds of knowledge, using them to perform a correspondingly varying range of tasks. The plan will, very approximately, parallel the historical development of understanding systems by starting with systems that incorporate modest amounts of knowledge and perform limited tasks, then proceed to systems that are relatively rich in knowledge and in the range of tasks to which they are applicable. The earlier and simpler systems will be treated briefly in the next three parts of this section; some examples of more recent and more elaborate systems are in the next three sections. The primary examples I shall develop in these latter sections are two systems, UNDERSTAND and THERMO, developed by my colleagues and myself, which have not previously been described in the AI literature. I will also compare these examples with other systems that have been reported in the literature.[2]

[2]No attempt will be made in this paper to survey the whole range of computer systems that understand. For leads to the literature of the subject, I would recommend Bobrow £ Collins (1975), Simon and

Parsing and Inferential Capabilities

One of the earliest applications of the term "understanding" to a computer program (perhaps the earliest) occurs in the title of R. Lindsay's chapter in Feigenbaum & Feldman (1963): "Inferential Memory as the Basis of Machines Which Understand Natural Language." In Lindsay's words (p. 221), "the program. . .parses sentences written in Basic English and makes inferences about kinship relations. To do this it constructs two types of complex structures in the computer memory, one corresponding to a sentence diagram of the sort produced by high-school students, the other corresponding to the familiar family tree." Performance of two kinds of tasks demonstrates the system's use of knowledge "whenever appropriate." When it is presented with a sentence like, "John is Mary's father," it can annex the information in that sentence to the genealogical chart of John and Mary's family. If it has also stored in that chart the fact that Joe is John's brother, it can then make a simple inference to answer a question like, "Name one of Mary's uncles."

Thus, to Lindsay, understanding meant encoding information from natural language input and answering questions about that information, including drawing inferences from it. Essentially, he identified understanding systems with a particular class of question-answering systems. This identification persisted through most of the 1960's. When the terms "comprehend" and "understand," which were not much employed during the middle of that decade, came into frequent use again (Weizenbaum, 1966; Simmons, Burger ε Schwarcz, 1968; Quillian, 1968), it was in reference to systems with these sorts of capabilities. In all of these systems, the knowledge they understand is "book larnin'." They read sentences; they store the information from them; they answer questions relating to this information, possibly drawing inferences in order to do so. But they know nothing of the real world to which the information purports to refer. If the BASEBALL program (Green, Wolf, Chomsky ε Laughery, 1961) can answer the question, "Where did the Red Sox play on July 7?", it is because it has been told the answer, and not because it remembers having been at the game.

The definition of understanding implicit in these early natural-language question-answering systems might be phrased:

> A question-answering system, S, understands knowledge, K, if it can answer natural-language questions about the explicit content of that knowledge, or questions involving (relatively simple) inferences from that knowledge.

The knowledge that the system understands falls in two categories. The first is knowledge of the syntax of the inquiry language, perhaps embodied in a program for analysing and generating grammatical sentences in that language together with lexical knowledge of word meanings. The second is substantive knowledge about the domain of inquiry, e.g., about some family tree in the case of Lindsay's system. These different kinds of knowledge will not always be segregated in different parts of the program structure; the boundary between syntax and semantics may be (and, in complex systems, generally will be) quite vague.

Understanding Intensional Meanings.

A different dimension of understanding is represented by programs that begin to appear at the end of the 1960's (Coles, 1967, 1972; Siklossy, 1968, 1972; Winograd, 1972). These programs accept natural language input, but they also are able to acquire information from an independent channel (simulated visual information in each case) about an outside world, and to relate the language to that world. We may call systems that communicate with such a real or simulated world quasi-robots.

Coles' program, for example, demonstrates its understanding by answering questions about a picture drawn with a light pen on a graphic display, and by parsing ambiguous sentences, selecting the particular parsing that corresponds to the picture. Thus, when required to answer "true" or "false" to the statement, "Each polygon smaller than a black triangle is a square," it arrives at its answer by inspecting the information in the picture to see what polygons, triangles, and squares are displayed in it, and whether these objects satisfy the conditions of the sentence. Coles' program, therefore, must have available intensional meanings for words like "polygon," "smaller," and so on, in the form of programs that will actually test whether a particular object, property, or relation in the visual representation fits the meaning. It must also be capable of matching these meanings against the corresponding natural language statements--a translational capability.

Siklossy's system has similar translational capabilities. It learns the vocabulary and syntax of a natural language by being shown paired sentences and "pictures" that the sentences describe, in the manner of the I. A. Richards books, English Through Pictures, German Through Pictures, etc.

The particular requirements of understanding in quasi-robots lead to a definition of the foltowing sort:

> A quasi-robot, S, understands language, L, if S possesses intensional definitions of expressions in L that enable it to determine what objects and events in an independently described world are denoted by these expressions.

Siklossy (1972), and Winograd (1972). For some of the more recent developments in the area of automatic programing, see Heidorn (1976). The particular examples discussed in the present paper, in addition to UNDERSTAND (Hayes & Simon, 1972) and THERMO (Bhaskar 6 Simon, 1977), were selected because of my familiarity with them, or because they illustrate specific points about the nature of understanding systems.

Self-Referential Meanings

Winograd's SHRDLU system combines these kinds of capabilities for understanding with a few others. Like the two systems just described, it must be considered a quasi-robot, for the world it knows is a synthetic one, and not the real one. SHRDLU both answers questions about its block world, and carries out commands to make changes in it. To do this, it needs the same forms of understandIng--of both language and its environment—as do the programs of Coles and SIklossy. But SHRDLU exhibits also some introspective forms of understanding that were only implicit in earlier problem-solving programs. Any program that makes use of goal and subgoal structures for its problem solving understands the motivations of its own actions, and uses that understanding to guide its search activities. SHRDLU is able also to access this information in response to a user's inquiry, and thereby to state reasons for its actions.

Contrast of Intensional with "Verbal" Understanding

The important distinction between understanding that involves the ability to apply intensional tests to "external" objects (I will resist the temptation to call it real understanding) and understanding that involves only verbal knowledge is illustrated by the following SHRDLU dialogue reported by Winograd (1972, p.11). The statements prefixed by a "+" are a user's assertions (to be added to SHRDLU's data base) or questions. The other statements are SHRDLU's replies:

+ I own blocks which are not red, but I don't own anything which supports a pyramid.

I UNDERSTAND.

+ Do I own the box?

NO.

Now although SHRDLU's answer to the question is quite correct, the system cannot be said to understand the meaning of "own" in any but a sophistic sense. SHRDLU's test of whether something is owned is simply whether it is tagged "owned." There is no intensional test of ownership, hence SHRDLU knows what it owns, but doesn't understand what it is to own something. SHRDLU would understand what it meant to own a box if it could, say, test its ownership by recalling how It had gained possession of the box, or by checking its possession of a receipt in payment for it; could respond differently to requests to move a box it owned from requests to move one it didn't own; and, in general, could perform those tests and actions that are generally associated with the determination and exercise of ownership in our law and culture.

Understanding Problem Solutions

In distinguishing the kinds of understanding that are incorporated in natural language question-answering systems (language-parsing and inferential capabilities) from the additional kinds that are required for quasi-robots and robots, we have by no means exhausted all the species of understanding. In this section, I shall discuss what it means to understand the solution of a problem; in the next section, what it means to understand the problem Itself. In both bases, as In previous sections, I will use existing AI systems to Illustrate the discussion.

The Gestalt psychologists have placed considerable emphasis upon the distinction between solving problems In "meaningful" and "meaningless" ways. We can use the familiar Tower of Hanoi puzzle to explicate what this distinction might mean. The Tower of Hanoi puzzle consists of three vertical pegs and N doughnut-shaped disks graded In size. At the outset, all the disks are arranged in a pyramid, from largest to smallest, on the first peg. The problem is to arrange them in a similar pyramid on the third peg. Only one disk may be moved at a time, and a disk may never be placed atop another that is smaller than it is. A pyramid of N disks can be transfered with a minimum *of* 2^N-1 moves.

The N-disk Tower of Hanoi problem can be solved by many different algorithms. I shall mention four.[3]

1. A rote algorithm simply consists of a sequence of 2^N-1 instructions corresponding to the successive moves. A different algorithm is required for each N.
2. The familiar recursive algorithm requires essentially three instructions: If the problem is to move N disks from Peg X to Peg Z, move N-l disks from Peg X to Peg Y, then the largest disk from Peg X to Peg Z, then N-l disks from Peg Y to Peg Z. This recursive algorithm works for any finite number of disks.
3. A means-ends algorithm avoids the necessity for retaining a goal stack. (a) If the problem is to move N disks from Peg X to Peg Z, find the largest disk not on Peg Z, and set up the goal of moving it. (b) If this disk can be moved to its target peg, move it, delete the goal, and go to (a). (c) Otherwise, delete the goal of moving it and find the largest of the disks that either lie above it or lie on its target peg. (d) Set up the goal of moving that disk to the remaining peg, and go to (b).
4. A Pattern algorithm. Suppose the problem is to move N disks from Peg X to Peg Z. (a) On odd moves, move the smallest disk; on even moves, the other movable disk. (b) If N is odd, move the smallest disk in the cycle, Z, Y, X, etc.; if N is even, in the cycle, Y, Z, X, etc. (These rules define the strategy uniquely, since there is only a single legal move for the "other movable disk.")

The last three algorithms are different from the first, rote, algorithm in making use of knowledge

[3]For a fuller and more formal description of these algorithms, see Simon (1975).

about the problem structure to encode the move strategy parsimoniously. It might be argued that algorithm (2) reveals the deepest "understanding" because It is very easy to prove that this algorithm constitutes a minimum-path solution of the problem; whereas that proof is somewhat more difficult for algorithms (3) and *(4)*. These comments, however, depart from our operational definitions of understanding; we must ask how the apparently greater depth of understanding exhibited by algorithm (2) can be detected in the overt behavior of the system.

In human psychology, the parsimony of algorithms for solving problems has a variety of behavioral consequences. Generally speaking, the more parsimonious the algorithm, the less the time required to learn it, the better it is retained in memory, and the wider the range of problem situations to which it can be applied (transfer).

Although the appropriate experiments have not been carried out for the Tower of Hanoi problem, experimental studies on other puzzle-like problems (Katona, 1940), together with theoretical considerations, make it fairly obvious that the rote algorithm would take much longer for a person to learn than the others. Similarly, experimental evidence (Katona, 1940) suggests strongly that algorithms like the first one would be less well remembered after a lapse of time than the others. It would be difficult, however, to interpret these differences as differences in the depth of understanding represented by the algorithms. They say nothing about whether the knowledge embedded in the respective algorithms will be applied whenever appropriate.

With respect to transfer, on the other hand, the differences in range of application of the knowledge are obvious. The rote algorithm, unlike the others, only works for a particular value of N; it will not transfer to problems that differ in even the smallest detail from the original one. At the other extreme, the means-ends algorithm will solve the problem from any initial starting distribution of the disks, and not just from the usual initial pyramid or from other distributions that lie along the solution path. In this important respect, it exhibits greater understanding than algorithms (2) and *(4)*, being capable of using its knowledge over a wider range of circumstances. From this example, we see how we can attribute different degrees of understanding to problem-solving programs, and how the degree of understanding is related to incorporating knowledge about the problems in the programs.

Those Tower of Hanoi programs--the means-ends and recursive algorithms—which incorporate problem-reduction techniques, also possess the basic capabilities for the kind of self-referential understanding that was illustrated earlier by the SHRDLU example. The means-ends and recursive algorithms can answer the question, "Why did you do that?" in the same way that SHRDLU answers it: by reporting the goal and subgoal structure that led to each move. By this test, the means-ends and recursive algorithms would exhibit understanding, the rote and pattern algorithms would not.

Understanding Problems

We have just seen how a problem-solving algorithm can be said to exhibit understanding of a problem. Usually, however, when we say that someone understands a problem, we do not mean that he has an algorithm that can solve it. It Is very easy, for example, to understand what problem is posed by Goldbach's conjecture: Prove (or disprove) that every even number can be expressed as a sum of exactly two primes. The problem itself is a very hard one, which no one has yet succeeded in solving. Hence, we cannot test anyone's understanding of Goldbach's conjecture by asking him to produce the solution. How can we test it?

Consider a language acquisition program that can encode the natural language strings that are input to it and store them in long-term memory. What is an appropriate way to encode a problem 1 ike Goldbach's conjecture? Suppose that the system already possesses some kinds of general problem solving capabilities--some version, say, of GPS. Then for the system to go to work on a problem, the problem will have to be encoded into a form which its problem-solving mechanisms can accept as input. That is to say:

> A problem-solving system, S, understands a problem, P, when it can encode the input instructions, t, that describe P into an internal representation, R, that is suitable as an input to the problem-solving subsystem of S.

Of course, if the problem-solving component of S is very weak in the face of problems like P, we may not be impressed by the depth of understanding achieved by the encoding. We say the understanding is deeper, the more efficient is the problem-solving subsystem of S for problems represented in format R.

Several cases need to be distinguished, on the basis of what the system already knows about problems of the class of the one newly presented to it. If the system already has a general scheme of representation for such problems, then understanding a new problem of the same kind may require little more than a parsing capability. The SHRDLU system, for example, deals with problems in a single block world, with a fixed representation. When it is instructed to "pick up a big red block," it needs only to associate the term "pick up" with a procedure for carrying out that process; identify, by applying appropriate tests associated with "big," "red," and "block," the argument for the procedure; and use its problem-solving capabilities to carry out the procedure. In saying "it needs only," it is not my intention to demean the capabilities of SHRDLU. It is precisely because the program already possesses stored programs expressing the intensions of the terms used in inquiries and instructions that its interpretation of those inquiries and instructions Is relatively straightforward

Those stored programs are the foundation of its understanding.

The situation is different for a problem solver that is expected to understand problems for which it does not already possess complete representations. In the remainder of this section, I wish to discuss what is involved in this case, and to describe a system, capable of handling simple situations of this kind, which has been constructed by J. R. Hayes and myself (Hayes ε Simon, 1974, 1977; Simon & Hayes, 1976). In this system, called UNDERSTAND, the boundary between the problem-understanding subsystem and the problem-solving subsystem will be sharper than it is in a system like SHRDLU, for the UNDERSTAND system has been constructed in ignorance of the problem domain with which it will have to deal. Consequently, the problem-understanding subsystem will have a more complicated task than just mapping the input language onto the intensions stored in a lexicon. It will also have to create a representation for the information it receives, and create meanings for the terms that are consistent with the representation.

In designing a system to understand problems, there is a tradeoff, then, between providing the system with prior knowledge of the problem domains it is expected to handle, and providing it with procedures for creating problem representations that eliminate the need for that prior knowledge. SHRDLU is an example of the former strategy, UNDERSTAND of the latter.

The UNDERSTAND Program

Consider a problem-solving system consisting of two components, UNDERSTAND and SOLVE, organized to operate serially. UNDERSTAND receives as its input a statement of a problem, written in natural language, and produces as its output a representation of the problem that is suitable as an input to SOLVE. SOLVE takes the latter input and undertakes to produce from it a solution of the problem. My concern here is with the design of UNDERSTAND, but in order to make that design problem definite, it is necessary to say something about SOLVE and the form in which it takes its inputs. I will assume that SOLVE is a variant of the General Problem Solver (GPS)[4]; that is, given a representation of a starting situation and a goal situation, it undertakes to transform the former into the latter. Its techniques for doing this comprise processes for detecting differences between pairs of situations, processes for finding operators relevant for reducing differences of each kind it can recognize, and processes for applying operators to the given situation in order to transform it into a new situation.

The basic mechanisms of GPS are independent of the particular problem it is asked to solve, or the domain to which that problem belongs. In order to apply itself to any given problem, however, it must be provided with: (1) means for representing situations in the space, S, of the given problem or class of problems, (2) a set of processes for detecting differences, D, between pairs of situations so represented (these processes then being functions from SXS to D), (3) a set of operators, 0, for transforming one situation into another (i.e., functions from S into S), (4) a function (table of connections) from differences into sets of operators which defines the subset of operators that is relevant to reducing each kind of differences, and (5) an ordering of the differences to determine priority in their elimination.

Several investigators have shown how GPS, given only a representation for problem situations (1) and a set of appropriately represented operators (3), can be augmented with relatively simple learning capabilities that enable it to build its own ordering of differences (5), its own table of connections (A), and even its own set of differences (2).[5] It will be assumed that the GPS under consideration here has been augmented in this way, so that the UNDERSTAND program need provide it only with the problem situation representation and the set of operators.

As a further specification of GPS input requirements, and hence of the output of UNDERSTAND, I will assume that the two programs will communicate in a list-processing language like LISP or SN0B0L. A situation will be represented by list structures, which may incorporate description lists (property lists).[6] The operators provided to GPS by UNDERSTAND will take the form of LISP (or SN0B0L) programs.

Consider, as a simple example, the Tower of Hanoi problem, which we have already discussed. We can associate with the symbol, SITUATION, a description list with attribute, PEGS. The value of PEGS can be the list of three pegs, PEGA, PEGB, and PEGC. With each peg, we associate the attribute, DISKS, whose value is the list of disks on that peg (see Figure 1).

Of course, this particular representation of the Tower of Hanoi problem is not unique. A situation could just as well have the attribute, DISKS, and each disk the attribute, PEG, whose value would be the name of the peg on which that disk was currently situated. (See Figure 2.)

[4]Ernst ε Newell, 1969; Newell ε Simon, 1972, Chapter 9.

These learning programs are described and discussed in Eavarone ε Ernst, 1970; Ernst ε Newell, 1969; and Newell, Shaw, ε Simon, 1960.

[6]A description list, or property list, is simply a device for associating with a symbol, s, sets of functors of the form A(s) = v, where A is an attribute of s and v the value of that attribute. Thus with a symbol, APPLE, we can associate a list 'COLOR RED SHAPE ROUND FLAVOR TART', say, to describe it.

Figure I. Tower of Hanoi Representation

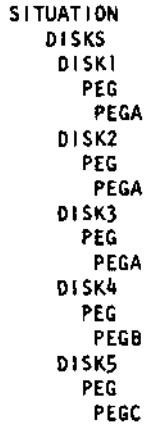

Figure 2. Alternate Tower of Hanoi Representation

The basic operator for the Tower of Hanoi problem changes the association of a disk from one peg to another, subject to certain conditions of "legality." The disk that is moved must be the smallest on its peg, and it must be smaller than any other disk that may already be on the target peg. For the moment, however, let us disregard the conditions and consider the bare move. Clearly, the algorithm required to carry out M0VE(DISK4,PEGB, PEGC), "move Disk 4 from Peg B to Peg C," will be different for the representation of Figure 1 than for the representation of Figure 2. In the former case, what is wanted is something like:

DELETE (DI SKA', DISKS, PEGB),INSERT(DISKA, DISKS, PEGC)

In the latter case, on the other hand, we want:

CHANGE(PEG,DISKA,PEGB,PEGC)

Even if we used a deletion and an insertion in the latter case, we would still arrive at a different program for the second than for the first representation:

DELETE (PEGB, PEG, DISKA) , INSERT (PEGC , PEG, DISK4)

We see that it is not enough for the UNDERSTAND program to construct from the problem instructions a list-structure representation of the space of problem situations, and an algorithm for moves. The algorithm must be designed to fit the particular representation that has been chosen.

Organization of the UNDERSTAND Program

To show how the UNDERSTAND program goes about its work, its operation on a simple task will be sketched. Figure 3 gives the written instructions for the Monster problem.

A MONSTROUS PROBLEM

Three five-handed extraterrestrial monsters were holding three crystal globes. Because of the quantum-mechanical peculiarities of their neighborhood, both monsters and globes come in exactly three sizes with no others permitted: small, medium, & large. The medium size monster was holding the small globe; the small monster was holding the large globe; and the large monster was holding the medium-size globe. Since this situation offended their keenly developed sense of symmetry, they proceeded to transfer globes from one monster to another so that each monster would have a globe proportionate to his own size.

Monster etiquette complicated the solution of the problem since it requires: (1) that only one globe may be transfered at a time, (2) that if a monster is holding two globes, only the larger of the two may be transfered, and (3) that a globe may not be transfered to a monster who is holding a larger globe.

By what sequence of transfers could the monsters have solved this problem?

Figure 3- Problem Instructions

A little consideration of this problem reveals that it is isomorphic to the three-disk Tower of Hanoi problem. Pegs have been mapped onto monsters and disks have been mapped onto globes. The somewhat more baroque language of the monster problem will perhaps underline some of the points I want to make.

The UNDERSTAND program operates in two main stages (see Figure A). First, it does a certain amount of syntactic and semantic parsing to transform the natural language sentences into somewhat more tractible form. I will call this part of the program the LANGUAGE algorithm. In the second stage, UNDERSTAND actually generates the representation and operators. I will call this the CONSTRUCT stage.

The Language Subsystem

There is little that is noteworthy about the LANGUAGE portion of UNDERSTAND. Quite conventional parsing methods are used. (In the first version of the program, these were largely home-grown. In a revised version now under construction, the machinery of Woods' parser (Woods, 1970) is being used.) The parsing routines will generally simply ignore any part of the text they cannot cope with. If these portions turn out to be inessential for deriving the representation and operators, no harm will

be done by the incompleteness of the product. In any event, the intended output of the program is not a parsing tree or a set of deep-structure kernel sentences, but the information that is essential for the construction processes.

UNDERSTAND
LANGUAGE
PARSING
IDENTIFICATION
(of objects
and relations)
CONSTRUCT
GENERATE REPRESENTATION
("Situation")
DESCRIBE REPRESENTATION
CONSTRUCT OPERATORS

Figure 4. Subroutine Structure of UNDERSTAND Program

The LANGUAGE processes undertake, in particular, to identify the sets of objects that appear in the problem statement (in our illustrative example, the two principal sets are the monsters and the globes) as well as the relations that are mentioned as holding among members of these sets (in the example the holding relation between a monster and a globe, and the transfering relation between a globe and a pair of monsters).

The LANGUAGE processes are also responsible for identifying which sentences in the problem text describe situations in the problem space (using such temporal cues as, "was holding," and such explicit designations as "this situation"); and for identifying sentences that describe the operators (the modal verb "may" is one of the cues employed).

The CONSTRUCT Subsystem

The CONSTRUCT program has three main subtasks: (1) to generate the representation of situations, (2) to construct a description of that representation, and (3) to find one or more appropriate operator algorithms in semantic memory, from which the problem operator can be built. The description of the representation is interpretable by the operator algorithms that are stored in semantic memory, thereby providing the information that is needed to adapt the operator to the particular representation that is chosen. By the use of this device, the problem, raised in a previous paragraph, of selecting the right form of the operator, is solved.

From a statement like "the medium-size monster was holding the small globe," the system infers that situations are describable in terms of a relation of the form: HOLDS(MONSTER,GLOBE). Notice that the system uses a many-sorted logic in which relations are described in terms of the types of their arguments. By a straightforward encoding, the relational information is now translated into a list-structure format like that depicted in Figure 1. Had the problem instructions been worded a little differently, they might have produced, instead, HELD.BY(GLOBE,MONSTER), in which case a list-structure resembling the format of Figure 2, instead of Figure 1, would have been created.

The DESCRIBE algorithm now takes the list structure named SITUATION, and creates a description of it, a symbolic string, which in the case of Figure 1 would read:

MONSTER MEMBER VALUE MONSTERS OF SITUATION

GLOBE MEMBER VALUE GLOBES OF MONSTER

Operators

There remains the portion of CONSTRUCT that designs the operators to represent the legal moves for the problem space. The accomplishment of this task is greatly facilitated by the fact that the situation is now represented by means of abstract and uniform list structures. All operators will be processes for changing list structures or making tests of the characteristics of list structures. Hence, the semantic information the system needs to have at its disposal, stored in its long-term memory, is (like the information needed by GPS) task-independent information: to wit, a basic set of primitive list-processing operations. It needs to have the capability of removing an item from a list, adding an item to a list, finding and changing a value on a description list, and so on. Since the primitives act in the abstract world of list structures, and not in the real world of physical and social actions, only a small finite set of them is needed. The system has no capability for adding to this set if it proves inadequate in any problem context.

The basic operators that are available in UNDERSTANDS semantic memory are classified according to the number and distribution of types of their arguments. Thus A(X,Y) would represent a process that takes two arguments that are possibly (but not necessarily) of different types. B(X,X,X) would represent a process that takes three arguments, all of the same type. The operator we require for the Monstrous Problem would be represented as C(X,Y,Y), for it takes three arguments, one of one type and two of another type. There are only a few basic operations on list structures that have this structure: change the value of the attribute on a list is one, transfer X from YI to Y2 is another, copy X from YI to Y2 is a third (differing from the second only in that it is non-destructive). Hence, the system requires only a relatively small amount of lexical information about the relational terms used in the problem statement (e.g., "transfer") in order to select the proper operator in memory.

In particular, verbs like "move," "transfer," "change," all map to the same algorithm. This algorithm, however, must be sufficiently sophisticated to carry out the particular action that is required by the specific representation that has been selected by CONSTRUCT. This requirement is met by writing the algorithms for the primitive processes so that they *are* executed interpretively under control of the description of SITUATION. Thus, for the description corresponding to the representation of Figure 1, TRANSFER(GLOBE,M0NSTER1, M0NSTER2) would find M0NSTER1 on the list of

monsters of SITUATION, find GLOBE on the list of globes of M0NSTERI, remove GLOBE from that list, find M0NSTER2 on the list of monsters of SITUATION and add GLOBE to the list of globes of M0NSTER2.

On the other hand, if the representation had been described as corresponding to Figure 2, the same TRANSFER operator, executed interpretively under control of this description, would first find GLOBE on the list of globes of SITUATION, then find the monster of GLOBE, and finally change the value of the monster from M0NSTER1 to M0NSTER2.

A relatively simple interpretive scheme with capabilities for recursion allows for indefinite variety in representation without unduly complicating the algorithms for the primitive operators. Most of the flexibility is embedded in the "find" process. The legality tests for moves *are* handled in the same way. In the problem at hand, "size" induces an ordering on the globes. This ordering is represented internally simply by storing the globes in a list described by the ordering dimension. To interpret a phrase like "the largest globe he is holding," the system first abstracts to an operator like: LARGEST(MONSTER(GLOBE)), which finds the appropriate globe. Such an operator is executed interpretively, in the first representation, say, by finding the monster associated with GLOBE, finding the list of globes of that monster, and (using the ordered list of globes) finding the last (i.e., "largest") of the monster's globes on that list.

UNDERSTAND can speed up its execution of operators, once the representation has been fixed, by retaining a trace of its interpretive execution of a function, then using that trace to compile out a specialized function applicable specifically to the given representation. With many of the tests for branches thereby eliminated, the compiled operators as might be expected, are an order of magnitude faster than the interpreted operators.

The Mechanisms of Understanding

To assess UNDERSTANDS understanding, several features of the program's structure call for additional comment, and for comparison with other programs having similar capabilities. The task performed by'the conjunction of UNDERSTAND with GPS is essentially an automatic programing task, or as Goldman, Balzer and Wile (1977) put it, a task of "translating a process description written in an informal, imprecise language with English-like semantics. . .into a process specification language with formal syntax and semantics." Its structure is similar in many respects to the structure of those authors' SAFE system (Balzer, Goldman, ε Wile, 1977). UNDERSTAND + GPS derives its general architecture from an earlier automatic programming scheme, the HEURISTIC COMPILER (Simon 1963, 1972).[7]

[7]Specifically, UNDERSTAND corresponds to the representation-generating portion of the HEURISTIC COMPILER that is described in Simon (1972), pp. 31-42.

A first characteristic of UNDERSTAND, dictated by the fact that it is to provide input to GPS, is that it treats the objects and relations mentioned in the problem instructions in an abstract way. All it needs to learn about them from the text is how to interpret them as list structures and operations on list structures. Since a programing language for operating on list structures needs only a relatively small number of basic operations, the semantics of UNDERSTAND, which is embedded in such operations, can be correspondingly lean.

The operations used by UNDERSTAND invite comparison with Schank's ACTs (Schank, 1975)- Although Schank emphasizes that there is no magic number of primitive processes required to express meanings, he argues that "it is possible to build an adequate system. . .using only eleven ACTs." (Schank, 1975, p.269). When we examine the list of ACTs that he proposes (pp.269-271), we see that he too goes the route of abstraction. At least three of the eleven ACTs are forms of TRANSFER, specialized by constraints on the types of objects that may appear as arguments. For example, the ACT that he calls ATRANS denotes the transfer of an abstract relation, while PTRANS denotes a change in physical location of an object. Both of these (as well as other transfer operations) would be handled in UNDERSTAND by the semantics of TRANSFER combined with tests on the types of its arguments. Hence, UNDERSTAND might be expected to handle a considerable range of problems with only eight or ten operat ions.

The UNDERSTAND program shows that this simplification of basic operations need not be interpreted as a claim that there are only a few kinds of actions in the real world. Rather, it is a claim that real world actions, of whatever kinds, can be represented internally by a few kinds of actions on list structures.

When the system has only a small number of primitive operators, then the task of mapping the actions named in the problem instructions onto these operators is correspondingly simplified. As was indicated in the last section, knowledge of the number of arguments in an operator and their distribution by types goes a long way toward identifying the operator in semantic memory that is appropriate for translating a particular action, although it will not do the whole job (e.g., it will not distinguish between a "copy" and a "transfer").

The system of Balzer, Goldman, and Wile (1977) carries this abstraction even a step further. In their system, all knowledge is handled in a uniform format as a set of assertions of relations, i.e., in some such form as (RELATI0N1 ARGUMENT1 ARGUMENT2). Clearly, the only operations needed for manipulating a data base in this form are operations for ascertaining the presence or absence of a relation, an operation for inserting a new relation, and an operation for deleting a relation. Since there is only one possible form of representation, their system does not require the capability, needed by UNDERSTAND, for describing its representation and modifying its operations (or interpreting

them) to fit that representation. The operations are fixed, once and for all. In the present implementation of the system, the price paid for this simplicity is a high cost of search for relations in the data base,[8] but I know of no demonstration that more efficient search methods could not be found without sacrificing the simplicity of representation. Such questions of processing efficiency lie outside the scope of this paper.

A second important characteristic of the UNDERSTAND program is that its capabilities for syntactic processing *are* relatively simple. It analyses the natural language instructions only to the extent it needs to in order to extract the objects and the relations that are relevant to the problem formulation. Perfect and unambiguous parsing of the natural language inputs is not usually essential, and semantic power is more important than syntactic sophistication in the language processor. This same conclusion has been reached by other designers of contemporary understanding systems. Brown ε Burt (1975, p.324) say, for example, of their SOPHIE system, which carries on dialogues relating to electronic circuits: "SOPHIE had to cope with problems such as anaphoric references, context-dependent deletions, and ellipses which occur naturally in dialogs. In fact handling these constructs seemed more important than building a system endowed with great syntactic paraphrase capabilit ies."

A third characteristic of UNDERSTAND is that design decisions made by one component of the program while processing one part of the problem text are accessible to other components of the program processing other parts of the problem text. Balzer, Goldman & Wile obtain this accessibility by their uniform representation of all relations. In the UNDERSTAND system, it is achieved by constructing descriptions of the representation which can be read and interpreted by UNDERSTAND's primitive operations. The algorithm that writes the descriptions is a self-referential procedure thatgathers information about the data structures being assembled by UNDERSTAND and describes them in a language readable by the system itself. Obviously, the program could easily be augmented to demonstrate this component of its understanding capabilities by answering questions of the form: "How have you represented problem situations in terms of list structures?"

Conclus ion

The foregoing discussion illustrates how a system can be designed to understand new problems--that is, to encode them so they will be accessible to a problem solver. In the UNDERSTAND system, this capability appears to be limited to problems that are abstract. We now return to a consideration of understanding in task domains that *are* relatively rich in semantic information.

R. Balzer, personal communication.

Understanding in Semantically Rich Domains

A large part of the current research effort on understanding systems is directed at the development of problem solving systems to operate in semantically rich domains. By a semantically rich domain I mean one in which a good deal of substantive information must be supplied to the problem solver, and be understood by it, in order for it to represent and solve, the problems. Among recent understanding programs that fit this description are the SOPHIE program of Brown & Burton (1975), whose domain of understanding is electronic circuits, the ISAAC program of Novak (1976), which deals with physics problems in the domain of statics, and a collection of programs, which I shall call THERMO, designed by R. Bhaskar and myself to generate problems in chemical engineering thermodynamics.[9] Each of these programs possesses the knowledge that is acquired in a portion of a college-level course, and uses this knowledge either to understand and solve natural-language problems (ISAAC), to generate problems in its domain (THERMO), or to work in tutorial mode with students (SOPHIE). Among the points of interest in programs of these kinds is that they begin to provide us with estimates of the amounts of semantic knowledge that are required to represent real-world domains, and that are acquired by students in their courses at school.

These programs resemble SHRDLU, in that almost all of their semantic knowledge about the problem domain is provided to them in the form of built-in or programmed-in data structures and algorithms stored in advance of the presentation of specific tasks. This is especially true of SOPHIE, which has to be supplied with explicit representations of the specific circuits it will later use in its tutorial interaction with students. While both ISAAC and THERMO are supplied with semantic knowledge about the components of the sorts of systems with which they deal, they have the additional capability of assembling those components into specific, instantiated situation descriptions. Thus, THERMO knows about processes, like heat exchange, devices, like ducts and pumps, and working substances, like steam and nitrogen. It can use this informat ion to generate, for example, a problem about heat exchange through a device where air enters at one temperature and pressure, and leaves at another (Figure 5). ISAAC knows about weights,

[9]Limitations of space prevent me from discussing speech understanding programs, which also make use of large amounts of stored information in performing their task, but belong to a very different species from the programs mentioned here. It would perhaps be less confusing to talk of "speech recognition" rather than "speech understanding", for the purpose of these programs is to find the graphemic transcriptions for oral natural language inputs, and not to understand those inputs. However, some recent speech recognition systems, like Hearsay-II try to attain semantic understanding to aid recognition. See, for example, Erman & Lesser (1975).

levers, pivots, ladders, walls, and the like, and can use this information to represent statics problems described, say, in terms of persons standing on ladders that are braced against walls.

```
THE WORKING FLUID OF A FLOW SYSTEM IS AIR.
THE WORK.DONE IS 592 LBF.FT/SEC.
THE INLET.TEMPERATURE IS 46.1 F..
THE OUTLET.VELOCITY IS 15-8 FT/SEC.
THE HEAT. INPUT IS 154.97 BTU/SEC.
THE INLET.AREA IS 4.6 SQ.FT..
THE INLET.PRESSURE IS 16.8 PSIA.
THE OUTLET.AREA IS 36 SQ.FT..
THE OUTLET.SPECIFIC.VOLUME IS 0.32 CFT/LBM..
WHAT IS THE OUTLET.PRESSURE?
```

Figure 5. Example of Problem Generated by THERMO

Each of these three programs, then, combines, in varying proportions, two kinds of semantic knowledge: (1) pre-stored general knowledge, and (2) specific knowledge obtained from the problem statement (in the case of THERMO, the latter is created by the program itself as it generates problems). Programs designed to operate in semantically rich domains can be classified not only with respect to the total amount of semantic information they use, but also with respect to the relative proportions of it they acquire through these two routes. SHRDLU and SOPHIE stand at one end of this continuum, relying mostly on pre-stored knowledge, while THERMO and ISAAC lie somewhere closer to the middle. If we were to place an abstracting program like UNDERSTAND on the same continuum, it would lie at the other end: all of the knowledge it uses is extracted from the problem instructions. I will return to this taxonomy later, for I think it has significant implications for the design of understanding systems with broader capabilities than the existing ones.

The Knowledge Content of THERMO

It will be useful to sketch out a bit more concretely the kinds of knowledge required by a system that is to understand problems in thermodynamics, and a possible method of storing that knowledge so that it will be accessible. The THERMO system is at present limited to steady-state flow problems in chemical 'engineering thermodynamics.[10] The information that THERMO possesses about devices, processes, and substances is stored in the form of schemas.[11] Schemas, in turn, are simply represented as description lists. A working substance, for example, has a name, upper and lower bounds for the values of its state variables (temperature, pressure, etc.), and equations of state that determine each of these state variables as a function of two of the others. A device is described in terms of its sequence of processes, and the working substances that are admissible for use with it. A process is described in terms of default values of variables, including variables assumed constant through the process.

In addition to its descriptions of working substances, devices, and processes, THERMO holds In memory the equations of conservation of energy and conservation of mass. The energy conservation equation is stored in a quite general form that can be specialized, for use in particular problems, by the omission of unnecessary terms. As already mentioned, equations of state are stored with each of the working substances, since these equations are different for different substances,

THERMO makes up thermodynamics problems by selecting (using random numbers when appropriate) a system that consists of a sequence of devices (possibly including cycles), and a sufficient set of values of input and output variables to determine the behavior of the system uniquely. THERMO knows that the values of output variables at each stage of a process must equal the input values at the next stage, and that the changes in values must zero out over any cycle. Various equation-counting procedures are available to the program so that the system will be neither under- nor over-determined. The program selects a particular variable or variables as dependent variables, selects values for a sufficient set of independent variables, and formulates a problem for the user of computing the values of these dependent variables.

[10] There is as yet no publised description of THERMO in the literature. Two components, ENTIR and REVIEW has been implemented, each capable of generating complete problems, but specialized along different dimensions. A more general program, combining and augmenting the capabilities of the existing components, has been designed but not implemented. An analysis of human problem solving in thermodynamics that provided part of the motivation for the design of the THERMO programs will be found in Bhaskar ε Simon, 1977.

[11] It is regrettable that Minsky has chosen to use the term "frame" instead of "schema" for this concept. I committed a similar sin by using "template" to refer to the schemas (of exactly the same kind) that I employed in the HEURISTIC COMPILER (1963, 1972). There are two substantial reasons for preferring "schema." First, the term has become well established since it was Introduced by the psychologist Bartlett in 1932, and has been widely used in this sense by both psychologists and researchers in artificial intelligence (e.g., Norman ε Rumelhart, 1975, pp.406-407). Second, Hayes and McCarthy (McCarthy 6- Hayes, 1968; Hayes, 1971, 1973) had already introduced the term "frame" into the artificial intelligence literature with an entirely different sense, in connection with the so-called "frame problem." Since using the term in two such disparate ways in the same literature can only cause confusion—and already has—"schema" should again become the standard usage. It would seem to be a matter of indifference whether "schemas," "schemata," or both, are used in the plural, since no confusion can result in either case.

Since, in the course of constructing a problem, THERMO also solves it, it holds in memory information that could be used for tutoring a student interactively, somewhat in the manner of SOPHIE. It could provide a student not only with feedback on the correct values of particular unknowns, but also about possible solution paths, i.e., the most convenient order in which to solve the equations in order to reach a conclusion. These tutorial capabilities have not yet been implemented.

The Frame Problem

The frame problem (Hayes, 1973) is the problem of keeping track of the consequences of performing an action or of making some alteration to a representation of a situation. Updating the representation of a complex situation as it undergoes successive modifications is a distinctly non-trivial task. We need to consider how the frame problem is handled in programs of the sorts we are discussing in this section.

The fundamental way in which THERMO's understanding exhibits itself is in its ability to assemble the information stored in individual schemas associated with devices, substances, and processes into descriptions of compound systems made up of these elements, and to formulate consistent problems (i.e., instantiations) for these systems. The basis for this understanding, in turn, lies in THERMO's means (1) for keeping track of equations and variables so that the system will be just determined, and (2) for equating input and output values of subprocesses to assure consistency, including consistency around loops.

One can identify similar mechanisms in SOPHIE, which has the capability of changing the values of one or more circuit parameters, and then calculating the resulting values of all circuit variables. Likewise, the ISAAC system is able to fit together components like levers, pivots, and weights into descriptions of a total problem situation, taking account of the constraints at junction points imposed by the physics of the situation (e.g., the resultant force at each junction must be zero).

The mechanisms in THERMO, SOPHIE, and ISAAC for coordinating system components and assembling problem situations from these components may be recognized as solutions of the frame problem for these systems. For, in addition to the descriptive information associated with each schema, information is needed that allows the effects of interactions between instantiated schemas to be taken into account. Notice that in no case is a general solution provided to the frame problem. In each case, the mechanisms for handling interactions derive from the conservation laws peculiar to the semantic domain under consideration--in SOPHIE, the circuit laws, in THERMO, laws of conservation of mass and energy, and in ISAAC, laws of static equilibrium.

In abstract systems, like UNDERSTAND and the other programs discussed in the previous section, the frame problem is handled by assuming that all interactions are explicitly mentioned in the problem formulation—that in the abstract representation of the situation, moves have no Implicit side effects. In the domains we are now considering it is the function of a theory of the domain—of electronic circuits, of statics, of thermodynamics--to predict the side effects. The frame problem is solved in each domain by incorporating in the program the capability of using the relevant scientific theory of that domain. Hence, we may say that:

> A system, S, understands a scientific theory, T(D), for domain D if it can use its knowledge of T to calculate the consequences, direct and indirect, of assumptions about situations and changes in situations in D.

A "correct'' theory of any body of phenomena allows inferences to be made from the values of certain variables to the values of others. The simplicity or complexity of the computations required to make these inferences will depend on the mathematical structure of the theory. If the theory takes the form of N linear algebraic equations in N variables, then the capability of drawing correct inferences means the capability of solving systems of N simultaneous linear equations. If the theory takes the form of a set of linear equations and inequalities, combined with a maximization condition, then the capability of drawing correct inferences means the capability of solving linear programming problems.[12]

We should not, therefore, expect to find something that would constitute a general solution to the frame problem. There are as many frame problems as there are forms of mathematical and logical systems, and to solve the frame problem for any such system is simply to have an efficient computational algorithm for the theory of systems of that kind. For systems operating in semantically rich domains, the frame problem is solved by incorporating in the system a theory of the domain, together with computational algorithms appropriate to a theory having that structure.

The Acquisition of Knowledge

It is time now to return to our earlier discussion of knowledge acquisition in systems designed to operate in abstract and semantically rich domains, respectively. One can distinguish two kinds of knowledge acquisition, which in the psychological literature are often called accommodation and assimllation, respectively (Moore ε Newell, 197'0 .[13] Accommodation involves the acquisition of

[12] For a further development of this point of view, see Simon, 1965, 1967, 1972. In these papers I argue that the attempt to deal with the frame problem by constructing modal logics is misguided, and suggest as the alternative essentially the procedure outlined here.

[13] The terms are due to Piaget (Piaget & Inhelder 1969), and are used mainly in the literature of child development.

new internal representations, or the modification of existing ones. Assimilation involves relating new information to representations that are already available and to information that is already stored in memory.

At a local level, the assimilation problem arises initially in the language parsing stage of acquisition from the problems of anaphoric reference. At a slightly more global level, it arises in a system like UNDERSTAND from the necessity of coordinating the representation it has created of situations with the algorithms it has associated with operators. In systems like SHRDLU and SOPHIE, which contain large amounts of pre-stored information, it arises in accessing the lexical information associated with terms in the input text.

Consider a system that begins processing a lengthy text without any substantive knowledge of the semantic content of that text. At the beginning, the system is faced with a task of accommodation, for the only meanings it can attach to the terms of the text are the meanings they acquire from the internal symbol structures that the system builds up to represent them. In this sense, all new knowledge begins as "abstract" knowledge. As the system proceeds through the text, however, more and more of the content depends for its interpretation upon information that has been presented earlier, and which has presumably been stored internally in some form. In order to process this new content appropriately, the system must rely less and less upon its capability for creating representations de novo (accommodation), and more and more on its ability to match new information with information stored in semantic memory (assimilation).

For simple assimilative tasks, the system needs knowledge-accessing capabilities like those possessed by SOPHIE or ISAAC. For more complex assimilative tasks, which require the discovery and use of metaphoric and analogic relations between old and new knowledge, understanding calls for powerful and elaborate matching capabilities like those proposed in my HEURISTIC COMPILER scheme, by Moore and Newell (1974 in their design for MERLIN, or by Kling (1971) in his scheme for theorem proving by analogy.

Out of this welter of possibilities, we begin to see directions for the further development of systems to operate in semantically rich domains: systems that would not have to be provided with their full complement of semantic knowledge at the outset, but that could gradually acquire knowledge of a domain from a continuous input of text, starting with procedures for handling abstract (i.e., novel) material, and gradually shifting the burden of processing to procedures for matching new knowledge onto knowledge previously stored--a kind of hybrid between UNDERSTAND-like systems, and ISAAC- or MERLIN-like systems. The basic components from which we can manufacture such systems are becoming increasingly well understood.

Summary

Our discussion of semantically rich domains has focused on the frame problem, and upon the need of understanding systems to have strong capabilities for assimilation and accommodation of new information. A solution of the frame problem for any complex domain with strong interactions among its components implies having a theory of the domain, and adequate computational means for calculating the theoretical implications of system changes.

To understand the information that is presented to it, a system operating in semantically rich domains needs both capabilities for assimilating information to internal structures it already has, and capabilities for accommodating internal structures to novel information. There now exist systems with some capabilities in each of these directions, and an obvious next direction of research is to try to combine them more effectively.

Conclusion: The Varieties of Understanding

In this paper I have argued that understanding is a relation among a system, one or more bodies of knowledge, and a set of tasks the system is expected to perform. The development of understanding systems over the past twenty years has been a kind of two-part fugue, in which the proposal of a class of tasks to be performed generates a set of knowledge requirements for the system; while the knowledge a system acquires, in turn, enlarges its capabilities for understanding new tasks. I have sketched out a whole sequence of task-knowledge pairs that represent, in a rough way, the historical course that understanding systems have followed from the origins of AI to the present (Figure 6).

Knowledge	Tasks
1. Syntax	Parse ε store sentences
2. Inference procedures	Answer questions
3. Intensional meaning	Act on real or simulated environment
4. Goal stacks	Give reasons for actions
5. Problem-solving algorithms	Solve problems
6. Representation generation	Represent novel problems
7. Domain-specific schemes (assimilation)	Understand semantically rich problems

Figure 6. The Varieties of Understanding

The knowledge available to early understanding programs was mainly knowledge of the syntax of English (or some other natural language), and their understanding was demonstrated by their ability to use that knowledge to encode and store in memory information presented in natural language, and to recover that information in response to natural language inquiries.

A second kind of knowledge that the programs possessed to varying degrees was knowledge of certain rules of inference, which enabled them to

derive information that was only implicitly presented to them, and to use this implicit information to answer a broader range of inquiries. This inferential capability was sometimes limited to a knowledge of rules of logical deduction, but increasingly incorporated also semantic knowledge about the problem domain.

Intensional definitions of classes of objects and relations constitute a third kind of knowledge that a system may be equipped to use to answer questions about a real or simulated environment, or to manipulate objects in that environment. In these terms, a system (quasi-robot) understands what a snark is if it can apply tests that discriminate snarks from non-snarks; it understands what "turn" means if it has knowledge in the form of programs that enable it to execute a turn in response to a command.

A fourth kind of knowledge, associated with yet another class of tasks for testing understanding, is knowledge about the reasons for actions. Whenever a system's program enables it to engage in means-ends analysis in order to select its actions, it can be programmed to retain knowledge about the goal hierarchies it creates, and to use that knowledge to answer "why" questions about its own behavior.

Knowledge of algorithms for solving problems represents a fifth basis for understanding. In the case of humans, we test this understanding by measuring the time and effort required to acquire the algorithm (an inverse measure), the ease of its retention, and the range of problems over which it can be applied. We may also test it by asking "why" questions about the steps in the algorithm. We judge understanding to be deeper, the more successful the problem solver is in explaining the reasons for his actions.

A sixth kind of knowledge, knowledge that enables a system to create a representation and operators for a new and unfamiliar problem, can be distinguished from knowledge of how to solve the problem. If the problem is genuinely novel to the system, then understanding requires the ability (accommodation) to abstract the sets of objects and relations the problem is "about," and to encode them in a form that makes them accessible to a system component that understands how to go about solving problems so encoded.

Novelty is a matter of degree. That is, a system may have had varying degrees of experience with problems of a given sort, and may have been provided with varying amounts of semantic knowledge about such problems. A system may require not only knowledge of how to abstract and encode a problem text, but also knowledge of how to map terms and sentences contained in that text on knowledge structures already stored in semantic memory (assimilation). Such a capability constitutes a seventh kind of knowledge that is required by the compleat understanding system.

We do not have any assurance that the list of Figure 6 is complete--that there are not other kinds of knowledge and other kinds of tasks to which it can be applied. Even within the boundaries of this list of problems, it Is not evident that the repertory of mechanisms that we have employed in our programs is sufficient for achieving understanding at human or near-human levels; indeed, I would suppose that most researchers in artificial intelligence would think that they are not. We have no reason to be dissatisfied, however, with the progress we have made in the past two decades toward understanding the processes of understanding. We can well echo the sentiments of David Hume, in the opening pages of his Enquiry Concerning Human Understanding:

> It becomes. . .no inconsiderable part of science barely to know the different operations of the mind, to separate them from each other, to class them under their proper heads, and to correct all that seeming disorder in which they lie involved, when made the object of reflexion and enquiry. . .And if we can go no farther than this mental geography, or delineation of the distinct parts and powers of the mind, it is at least a satisfaction to go so far; and the more obvious this science may appear (and it is by no means obvious) the more contemptible still must the ignorance of it be esteemed, in all pretenders to learning and philosophy.

Acknowledgements

This research has been supported by Public Health Service Grant MH-07722 from the National Institute of Mental Health, Department of Health, Education, and Welfare, by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F44620-73-C-007M which is monitored by the Air Force Office of Scientific Research, and by a grant from the Sloan Foundation.

References

Balzer, R., Goldman, N. ε Wile, D. (1977) Informality in Program Specification, RR-77-59. University of Southern California, Information Science Institute, Marina Del Rey, California.

Bartlett, F. C. (1932) Remembering. Cambridge: Cambridge University Press,

Bhaskar, R. ε Simon, H. A. (1977) Problem solving in semantically rich domains: An example from engineering thermodynamics. Cognitive Science 1: in press.

Bobrow, J. S. ε Collins, A. (eds.) (1975) Representation and Understanding. New York: Academic Press.

Brown, J. S. ε Burton, R. R. (1975) Multiple representations of knowledge for tutorial reasoning in J. S. Bobrow ε A. Collins (eds.) Representation

and Understanding. New York: Academic Press, 1975, P. 311-350.

Coles, L. S. (1967) Syntax Directed interpretation of Natural Language. Doctoral dissertation, Carnegie-Mellon University.

Coles, L. S. (1972) Syntax directed interpretation of natural language, in L. Siklossy ε H. A. Simon (eds.) Representation and Meaning. Prentice-Hall, Inc., 1972, p. 211-287.

Eavarone, D. S. ε Ernst, G. W. (1970) A program that discovers good difference orderings and tables of connections for GPS. Proceedings of the 1970 IEEE Systems Science and Cybernetics Conference! New York: IEER, 1970, p. 226-233.

Elithorn, E. ε Jones, D. (eds.) (1973) Artificial and Human Thinking. San Francisco: Jossey-Bass.

Erman, L. & Lesser, V. (1975) A multi-level organization for problem solving using man, diverse, cooperating sources of knowledge. Advance Papers of the Fourth IJCAI, p. 483-490.

Ernst, G. W. & Newell, A. (1969) GPS: A Case Study in Generality and Problem Solving. New York: Academic Press.

Feigenbaum, E. A. ε Feldman, J. (1963) Computers and Thought. New York: McGraw-Hill.

Goldman, N., Balzer, R. ε Wile, D. (1977) The Inference of Domain Structure from Informal Process Descriptions.Unpublished technical report, University of Southern California, Information Sciences Institute, Marina del Rey, California.

Green, B. F., Wolf, A. K., Chomsky, C. ε Laughery, K. (1961) Baseball: An automatic question answerer. Proceedings of the Western Joint Computer Conference, p. 219-224.

Gregg, L. W. (ed.) (1974) Cognition and Knowledge. Potomac, Md.: Lawrence Erlbaum Associates.

Hayes, J. R. ε Simon, H. A. (1974) Understanding written problem instructions, in L. W. Gregg (ed.) Cognition and Knowledge. Potomac, Md.: Lawrence Erlbaum Associates, 1974, p. 167-200.

Hayes, J. R. ε Simon, H. A. (1977) Psychological differences among problem isomorphs, in G. Potts (ed.) Indiana Cognitive Symposium. Potomac, Md.: Lawrence Erlbaum Associates, forthcoming.

Hayes, P. J. (1971) A logic of action, in B. Meltzer and D. Michie (eds.) Machine Intelligence 6. Edinburgh: Edinburgh University Press, 1971.

Hayes, P. J. (1973) The frame problem and related problems inartificial intelligence, in A. Elithorn and D. Jones (eds.) Artiflcial and Human Thinking. San Francisco: Jossey-Bass, 1973. P. 45-59.

Heidorn, G. E. (1976) Automatic programming through natural language dialogue: A Survey. IBM Journal of Research and Development 20: 302-313.

Katona, G. (1940) Organizing and Memorizing. New York: Columbia University Press.

Kling, R. E. (1971) A paradigm for reasoning by analogy. Proceedings of the Second IJCAI. London: British Computer Society.

Lindsay, R. K. (1961) Toward the Development of Machines Which Comprehend. Unpublished doctoral dissertation, Carnegie-Mellon University.

Lindsay, R. K. (1963) Inferential memory as the basis of machines which understand natural language, in E. A. Feigenbaum ε J. Feldman (eds) Computers and Thought. New York: McGraw-Hill, 1963, p. 217-236.

McCarthy, J. & Hayes, P. J. (1968) Some philosophical problems from the standpoint of machine intelligence, in B. Meltzer ε D. Michie (eds.) Machine Intelligence 4. Edinburgh: Edinburgh University Press, 1968, p. 463-502.

Meltzer, B. ε Michie, D. (eds.) (1968) Machine Intelligence 4. Edinburgh: Edinburgh University Press.

Meltzer, B. ε Michie, D. (eds.) (1971) Machine Intelligence 6. Edinburgh: Edinburgh University Press.

Minsky, M. (ed.) (1968) Semantic Information Processing. Cambridge, Mass.: M.I.T, Press.

Moore, J. ε Newell, A. (1974) How can MERLIN Understand?, in L. W. Gregg (ed.) Cognition and Knowledge. Potomac, Md.: Lawrence Erlgbam Asoociates, 1974, p. 201-252.

Newell, A. (1963) Learning, generality, and problem solving. Proceedings of the IFIP Congress 62: 407-412.

Newell, A., Shaw, J. C. ε Simon, H. A. (1960) A variety of intelligent learning in a general problem solver, in M. C. Yovits ε S. Cameron Self-Organizing Systems. New York: Pergamon Press, 1960, pp. 153-189.

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

Norman, D. A. ε Rumelhart, D. E. (eds.) (1975) EXPLorations in Cognition. San Francisco: W. H. Freeman

Novak, G. S., Jr. (1976) Computer Understanding of Physics Problems Stated in Natural Language. Technical Report NL-30, Department of Computer Sciences, University of Texas, Austin, Texas.

Piaget, J. ε Inhelder, B. (1969) The Psychology of the Child. New York: Basic Books.

Potts, G. (ed.) (1977) Indiana Cognitive Symposium. Potomac, Md.: Lawrence Erlbaum Associates, forthcoming.

Quillian, M. R. (1968) Semantic memory, in M. Minsky (ed.) Semantic Information Processing. Cambridge, Mass.: M.I.T. Press.

Rescher, N. (ed.) (1967) The Logic of Decision and Action. Pittsburgh: University of Pittsburgh Press.

Richards, I. A. (1950's) English Through Pictures, Hebrew Through Pictures, etc. New York: Pocket Books.

Schank, R. C. (1975) The structure of episodes in memory, in. D. G. Bobrow 6 A. Collins (eds.) Representation and Understanding. New York: Academic Press.

Siklossy, L. (1968) Natural Language Learning by Computer. Unpublished doctoral dissertation, Carnegie-Mellon University.

Siklossy, L. (1972) Natural language learning by computer, in H. A. Simon ε L. Siklossy (eds.) Representation and Meaning. Englewood Cliffs, N. J.: Prentice-Hall.

Simmons, R. F., Burger, J. F. ε Schwarcz (1968) A computational model of verbal understanding. AFIPS Conference Proceedings, Fall Joint Computer Conference,33: 441-456.

Simon, H. A. (1963) The Heuristic Compiler. RM-3588-PR. The RAND Corporation, Santa Monica, California.

Simon, H. A. (1965) The logic of rational decision. British Journal for the Philosophy of Science 16: 169-186.

Simon, H. A. (1967) The logic of heuristic decision making, in N. Rescher (ed.) The Logic of Decision and Action. Pittsburgh: University of Pittsburgh Press, 1967, p. 1-20.

Simon, H. A. (1972) The heuristic compiler, in. H. A. Simon ε L. Siklossy (eds.) Representation and Meaning. Englewood Cliffs, N. J.: Prentice-Hall, 1972, p. 9-43.

Simon, H. A. (1975) Functional equivalence of problem-solving skills. Cognitive Psychology 7: 269-288.

Simon, H. A. ε Hayes, J. R. (1976) The understanding process: Problem isomorphs. Cognitive Psychology 8: 165-190.

Simon, H. A. ε Siklossy, L. (eds.) (1972) Representation and Meaning. Englewood Cliffs, N. J.: Prentice Hall.

Weizenbaum, J. (1966) ELIZA—A computer program for the study of natural language communication between man and machine. Communications of the ACM 9: 36-45.

Winograd, T. (1972) Understanding Natural Language. New York: Academic Press.

Woods, W. A. (1970) Transition network grammars for natural language analysis. Communications of the ACM 13: 591-606.

Yovits, M. C. £ Cameron, S. (eds.) (1960) Self-Organizing Systems. New York: Pergamon Press.