NOTICE WARNING CONCERNING COPYRIGHT RESTRICTIONS:
The copyright law of the United States (title 17, U.S. Code) governs the making of photocopies or other reproductions of copyrighted material. Any copying of this document without permission of its author may be prohibited by law.

HOW CAN MERLIN UNDERSTAND?

J. Moore and A. Newell

November 15, 1973

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pensylvania

This paper will appear in L. Gregg (ed.), Knowledge and Cognition, Lawrence Erlbaum Associates, Potomac, Md., 1973. This research was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F44620-70-C-0107) which is monitored by the Air Force Office of Scientific Research.

This paper addresses the question: "How is it possible to understand?". The source of knowledge for what we have to say is a long-standing research effort to produce an understanding program called Merlin. That effort has been one primarily in artificial intelligence and thus this paper will also reflect primarily artificial intelligence concerns. Its appropriateness in a psychological context rests on a view that artificial intelligence is simply part of theoretical psychology (Newell, 1970). Enough evidence for this general view exists that it need not be labored here (e.g., Hunt, 1971; Newell and Simon, 1972).

Since Merlin has not been reported extensively we will start by providing some historical background. This will lead to posing the problem of understanding in a particular way, namely as a series of design issues that must be met by any understanding program. We will briefly illustrate these issues by means of current work in artificial intelligence. What little can be said about the issues from current data in psychology will become clear in passing. With these issues as background we will then discuss Merlin and the design decisions that characterize it (including those still open). We will summarize by attempting to answer how it is that we expect Merlin to understand.

SOME HISTORY OF MERLIN

Merlin was originally conceived (with the name CSA, standing for almost nothing) in 1967 out of an interest in building an assistance-program for a graduate course in artificial intellience.[i] The task was to make it easy to construct and play with simple, laboratory-sized instances of artificial intelligence programs. Because of our direct interest in artificial intelligence, the effort transmuted into one of building a program that would understand artificial intelligence -- that would be able to explain and run programs, ask and answer questions about them, and so on, at some reasonable level. The intent was to tackle a real domain of knowledge as the area for constructing a system that understood.

Artificial intelligence might appear to be too advanced a domain of knowledge. On the contrary, it has some properties that make it quite attractive. It involves discrete symbolic structures, thus avoiding the continuous three-dimensional world of time and forces. There is little highly developed theory in artificial intelligence, so that much of its knowledge is embedded in programs, which exemplify various important concepts and issues. Thus, "understanding artificial intelligence" can be replaced by "understanding artificial intelligence programs" with a consequent gain in specificity of the domain of knowledge. The entire project was cast as one of intelligent CAI.

A memory structure was developed (of which much more later) and an extended attempt was made to explore the properties of this structure (Moore, 1971). The high point of that study was the representation of a version of LT (an early theorem proving program for the propositional calculus) in this generalized memory structure, such that the system (by then called Merlin 1) could put it all together and prove trivial theorems. The memory structure (which might be viewed in one way as a semantic net) was highly unstructured so that the problems addressed through these first years were basic ones of operating within such a generalized structure and were not at all ones of CAI. However, work was done at the same time on how to represent the programs of artificial intelligence in a way appropriate to being understood (Newell, 1969).

The goals of Merlin require progress on many of the central problems in artificial intelligence (though instantiated in a particular guise). We settled down to explore these as a basis for designing a full-bodied Merlin. We put together a Merlin 2, a Merlin 2.5 and currently are working on a Merlin 3. These are not to be looked at as complete programs for an understanding system, but as explorations of various facets of the design. Many of these facets will come out below in discussing one or another aspect of an understanding system.

Meanwhile (since 1967 was some time ago) other developments in artificial intelligence programs have occurred. Most notable from the standpoint of Merlin have been the planning languages, Planner (Hewitt, 1969, 1972) and QA4 (Rulifson, Derkson and Waldinger, 1972), along with Winograd's (1970) program realized in part in Microplanner. Also important were the continued developments of semantic nets (e.g.,

[i] The original group included Richard Young in addition to the present authors.

Quillian, 1968, 1969; and Carbonell, 1970) and related conceptual structures (Shank, 1972). We have needed to understand Merlin in relation to these developments and the underlying structure of this paper reflects that attempt.

THE NATURE OF UNDERSTANDING

Any attempt to analyse directly the nature of understanding must be seen as a preliminary and approximate venture. A general lesson of work in artificial intelligence is to lean heavily on the analysis of the structure of programs that accomplish specific tasks, and to treat less seriously general analyses of the nature of intellectual functions. Still a certain amount of framework is useful. Let us stipulate, then, that we wish to construct an understanding program. How shall we comprehend our task?

The Essence of Understanding

Let us attempt to state directly what we might mean in saying of subject S that he (or it) understands. First, we take understanding to refer to knowledge, not to things directly. To say that S understands object or activity X is elliptical for saying that S understands knowledge *about* X. The shift leaves hanging the ambiguity of what knowledge is understood. Thus, to artificial intelligence is associated an imprecisely defined domain of knowledge and to understand artificial intelligence is to understand some, but not necessarily all, of that knowledge.

Second, we take the ultimate criterion of understanding of knowledge to lie in its use:

S understands knowledge K if S uses K whenever appropriate. (1)

This might seem vague, invoking terms such as "use" and "appropriate", which appear to be no way better defined than "understand" itself. In fact, most of the concepts in statement (1) can be successfully explicated. The vagaries in applying the statement to specific cases lie in lack of empirical knowledge about the system under comment or in lack of analytical ability.

The basis for these assertions lies in the analysis of computer programs and their behavior. The knowledge that a program has is defined in terms of an agreed-upon encoding. This can be done successfully even when the encoding is in the structure of the program, rather than the content of particular data expressions in a memory. Whether a program has or has not a certain bit of knowledge on a particular occasion can be settled directly by an examination of its code and data structures. A program uses knowledge in the production of some external behavior when the representation of that knowledge forms part of the processing sequence for producing the behavior. The complexities of programs -- their size, conditionality, self-modifiability, and capability for irrelevant processing -- all complicate the analysis, but do not change its character.

The notion of appropriate can also be given precise statement. We posit of certain information processing systems that they have goals. This may be an externally imposed stipulation, though it can be grounded in the agreed-upon interpretation of the system's internal structure (that certain data structures operate as goals). Whether behavior is appropriate is determined by whether it serves to obtain the goal (or goals). This determination requires an analysis of the task environment and, from it, an objective determination of the courses of action that could possibly serve to obtain the goals. Given the objective analysis, it can be further determined whether the specified knowledge could have been used to produce one of these behavior sequences. A complication in the analysis is that generally no single item of knowledge is sufficient to produce goal-obtaining behavior. Thus the appropriate use of an item of knowledge is always against the background of a larger collection of knowledge that is sufficient to produce the total behavior. This implies a competence model of the system, which characterizes capacities solely in terms of total knowledge. The entire analysis is complex, with distinct possibilities for error. But there does not seem to be conceptual difficulties with it.

As noted, the claims that all these characterizations and analyses can be carried out rests in part on the everyday practice in computer science and artificial intelligence in analysing computer programs. It receives strong confirmation in the intensive and successful work over the last five years in program verification (see London 1972). The task of formally verifying that a program accomplishes a given process or possesses specified properties (e.g., it will terminate) requires formalization of most of the above scheme. These techniques have grown up in a situation where the internal structure (both data and program) of the information processing system is completely known. When we turn to the analysis of human behavior, we are in the position of conjecturing the internal structure. As such, additional sources of error open up -- in the inputation of goals, of knowledge, etc. But these complications do not affect the basic adequacy of this concept of understanding.

The above account takes understanding as a step-function: the system either understands or it does not; if it does then it always uses the knowledge when appropriate. Clearly we need to augment statement (1) with another:

Understanding can be partial, both in extent (the class of appropriate situations in which the knowledge is used) and in immediacy (the time it takes before such understanding can be exhibited). (2)

The two qualifications are directly related: the more time we allow, the more understanding the system may be able to demonstrate, given that it has the knowledge. However, the relationship is not simple and in artificial systems especially, knowledge is likely to be bound into structure in ways that make it available only in limited situations, independent of the amount of processing allowed.

The above analysis seems to us essentially correct -- that one need not seek further for an explication for the intellectual function of understanding. By itself, however, it does not help much in developing an understanding program such as Merlin. Part of the difficulty is in the lack of any good notion of extent. To this matter we now turn.

The Extent of Understanding.

If the study of understanding is approached in the same way as the study of other intellectual functions in artificial intelligence, a task environment is sought such that performance in the tasks of the environment indicates that the program understands. Thus, we generate chess playing programs, integration programs, particular puzzle solvers, etc. For understanding, this strategy does not quite work. The issue is the requirement for diversity of task. To understand is to be able to reveal that understanding in use in the context of any task, providing that other requisite understanding is available. Diversity of task, not sheer numerosity, is clearly the important consideration.

To our knowledge there exists currently no satisfactory approach to a taxonomy of tasks by which to assess diversity. The ultimate source of difficulty in obtaining such a taxonomy undoubtedly comes from the requirement that it reflect the structure of the problem solvers who are to perform the tasks. There is no trace at the moment of such a theoretically-based taxonomy. There are, however, two distinct approaches that should be mentioned, one based on natural language, the other based on the use of function terms.

It is no accident that much work in artificial intelligence on understanding systems is simultaneously an effort in natural language understanding (from Raphael, 1968 to Quillian, 1969, to Winograd, 1971). To trade on a metaphor: Take care of the language and the problems will take care of themselves. Put another way, natural language encompasses an immense and immensely diverse space of problems, though one that is unknown and uncharacterized in any independent way. Therefore, a problem solver that understood natural language would seem to be capable of being posed problems of sufficient diversity for assessing understanding just by means of language, without prior analysis or commitment to a task environment. Some of the surface appeal of programs such as SIR (Raphael, 1968) is exactly this: you speak various facts to it of your own choosing, thus creating your own microworld, and then pose problems within that world.

There are difficulties with this view of course. Understanding of task domains may be a precondition for understanding linguistic utterances about them -- simply reversing the aphorism above. Large classes of problems cannot be posed via language, e.g., those involving direct interaction with the environment. Linguistic analogues of physical problems (to give them an imperfect name) already commit to encodings of the environment, which may constitute an essential component of the problem. Nevertheless, the use of natural language provides one approach to describing task diversity, though it does so by finessing much of the problem.

The second approach is to obtain a set of function terms that appear to span all tasks. By a function term we mean a highly abstract term, usually a verb, that contains a means-ends notion. "Support" is a good function term, so is "Comprehend," though the means-ends orientation is somewhat more subtle. No good theory of function terms exists, which describes how they describe human intellectual activity or how they might perform similarly in artificial systems (however, see Freeman and Newell, 1971, for some discussion). But collections of function terms seem capable of spanning entire spaces of possibilities without providing an operational definitions of the space.

A good example of a functional taxonomy can be found in the work in the early fifties of a Committee of College and University Examiners to put together a taxonomy of educational objectives (Bloom, 1956).[i] Figure 1 gives the taxonomy in outline, though without the supporting definitions and documentation developed by the Committee. The taxonomy, though developed intuitively, was tested both on many existing statements of educational objectives and (more important) on its ability to classify large numbers of test items from existing tests of mental abilities.

The claim of such a taxonomy is that it comprehends all mental activities or skills without positing an underlying structural space in which these can be rigorously defined. Function terms, such as "analysis" and "comprehension", cover an indefinite set of situations. The very strength of such functional decompositions (to cover without being precise) also constitutes their main disadvantage from our current view (namely, to understand what it is to understand). For there is nothing in the taxonomy that helps discern the attributes of an understanding system. For instance, one hardly wishes to create an understanding system that has components corresponding to the separate domains of the taxonomy -- that is, an analysis faculty, a synthesis faculy, an evaluative faculty, etc. It can conceivably be used evaluatively (as Stu Card is doing) to organize the assessment of the diversity of the task domains of a given understanding system. But in its present state of development it does not seem capable of doing much more.

The rather unsatisfactory state of describing spaces of diverse tasks makes unattractive the principal methodological strategy of initially specifying the task domain and then programming to meet the demands of that domain. We turn then to an alternative strategy.

Design Issues

One way to reveal the nature of understanding systems is to ask what design decisions they incorporate with respect to a set of key issues. What are these issues? Like the taxonomy of Figure 1 they will be intuitively defined and will in general be described in functional terms. Unlike the taxonomy, the functions will correspond to a view of how an understanding system must be structured and what mechanisms are required to solve the problems an understanding system must face. They can reflect both difficulties and opportunities that arise from pervasive features of task environments or from the nature of information processing as we currently understand it. Another way to look at such a list is to ask what one wants to know when confronted with a new understanding system, so as to assess its distinctive contributions.

Figure 2 presents the list of design issues. They fall under eight main heads. Within some of these there are listed a series of subissues. These issues are not meant to be exhaustive, but rather those that have some bearing on the particular questions to be raised in this paper.

Let us run briefly over the list in the abstract, though it is partly self-

[i] We are grateful to Stu Card for pointing out the relevance of this work to us; he is making a much more thorough investigation of the use of such a taxonomy in the study of understanding systems.

- 1.00 KNOWLEDGE
    - 1.10 Knowledge of specifics
        - 1.11 Knowledge of terminology
        - 1.12 Knowledge of specific facts
    - 1.20 Knowledge of ways and means of dealing with specifics
        - 1.21 Knowledge of conventions
        - 1.22 Knowledge of trends and sequences
        - 1.23 Knowledge of classifications and categories
        - 1.24 Knowledge of criteria
        - 1.25 Knowledge of methodology
    - 1.30 Knowledge of universals and abstractions in a field
        - 1.31 Knowledge of principles and generalizations
        - 1.32 Knowledge of theories and structures

- 2.00 COMPREHENSION
    - 2.10 Translation
    - 2.20 Interpretation
    - 2.30 Extrapolation

- 3.00 APPLICATION

- 4.00 ANALYSIS
    - 4.10 Analysis of elements
    - 4.20 Analysis of relationships
    - 4.30 Analysis of organizational principles

- 5.00 SYNTHESIS
    - 5.10 Production of unique communication
    - 5.20 Production of a plan, or proposed set of operations
    - 5.30 Derivation of a set of abstract relations

- 6.00 EVALUATION
    - 6.10 Judgments in terms of internal evidence
    - 6.20 Judgments in terms of external criteria

Figure 1: Taxonomy of Intellectual Functions

REPRESENTATION: What internal representations of knowledge are used?

> Scope Issue: What knowledge can be represented?
>
> Grain Issue: What sorts of partial knowledge can be represented?
>
> Multiple Representations Issue: How does the system capitalize on the fact that ease of knowledge use is a strong function of how it is represented?

ACTION: How does system convert knowledge into behavior?

ASSIMILATION: How does the internal structure of the system make contact with the external task environment?

ACCOMMODATION: How does external structure become converted into new internal structure?

DIRECTIONALITY: How does the system use its knowledge appropriately for the task?

> Keep-progressing Issue: What do you do when you don't know what to do?

EFFICIENCY:

> Interpreter Issue: Inserting interpreters degrades performance multiplicitively.
>
> Immediacy Issue: Certain general methods (mostly involving search) are precluded as the requirement for immediate response increases.
>
> Formality Issue: Formalism introduces inefficiency by forcing uniformity of processing.

ERROR: How does the system cope with the ubiquituous existence of error?

> Frame problem: When the world changes, once-true encoded knowledge becomes false.

DEPTH OF UNDERSTANDING: How is knowledge brought to bear whenever it is appropriate?

> Access Issue: Does the system have access to all implications of the knowledge cast in the representation?

Figure 2: List of Design Issues

explanatory. All information processing systems have knowledge by virtue of an internal REPRESENTATION. There may be more than one such representation; indeed most systems split their knowledge between a data representation and a program representation. Thus the first thing to know about an understanding system is the set of representations it uses.

Several issues arise about representations. The first is that of *scope*: what knowledge can be represented, what cannot. The general question here is easily settled, for with only a few constructs, a sort of conceptual completeness is obtained. These constructs are those of the first order predicate calculus: propositions formed from predicates by conjunction, disjunction and negation, with the predicates themselves dependent on variables ranging over a given domain of discourse and subject to universal and existential quantification. With this much, realized in any of many forms, all of mathematics and physical science is formalizable, and by extension most anything else one can become definite about. This claim is of necessity informal and really amounts to showing that apparently more expressive mathematial systems (especially set theory) can be formulated within first order calculus. It need not detain us, except to note that questions of the completeness of expressive power do not in general discriminate among current understanding systems (though a few early ones were highly limited).[i]

The second issue is that of *grain*. One way of viewing a problem solver is that he has a developing state of knowledge about his problem: each action that he takes generates some more partial knowledge. To be able to add this partial knowledge to his current total state requires a representation that can encode it. If the stock of concepts is not adequate, the problem solver is forced to approximate the knowledge in some way. The net effect of this approximation is inevitably to introduce the possibility of error, which shows up in many problem solvers as search, with its resultant combinatorial explosion.

An example will help. In a cryptarithmetic puzzle (Newell and Simon, 1972) the problem solver must ultimately assign digits to variables (e.g., R will be 0, l, ... 9). If at a certain point the problem solver infers that R must be odd, then if he has the concept of odd he can augment his knowledge state appropriately. However, if all he has is the ability to represent the integers, then he must overcommit, so to speak, and indicate that R is 1 or 3 ... . Even if he has the concept of disjunction, so he can remember that R is 1v3v5v7v9 this may not avoid the consequences, if his only way of dealing with assignments is in terms of digits (i.e., he does not have the equivalent of odd+odd = even). The combinatorial explosion arises becauses enumeration of the alternatives in partial knowledge is the general way to deal with being unable to represent (and manipulate) a partial state directly.

The final representation issue is that of *multiple representation*. This rests on the common observation that how a problem is represented is often a major aspect of solving it. Thus finding an appropriate representation for a problem should be an

[i] An important special issue is the extent to which the system can represent its own strategies of action to permit processing other than evocation. We will not be able to deal with this here.

important activity. Unfortunately, the current art is not up to handling multiple representations within the same system. Thus, the import of this issue is whether a proposed understanding system has any devices for capitalizing on multiple representations.

The second main design dimension, ACTION, concerns the fundamental problem of how a system derives behavior from the knowledge that is encoded statically in its structure. A range of options are available, from the application of inference rules, to the use of language interpreters (note that a program is as much a static representation of knowledge as a data base). Behavior, refers not only to ultimate external behavior, but also to internal behavior used mediately in generating final behavior.

The third dimension, ASSIMILATION, reflects another fundamental problem for any understanding system. At a given point in time all that the system knows is encoded somehow in its internal structure. An encounter with a task in its environment brings the system into contact with another body of structure. How are the two to make contact? If the external environment were of only small variety with respect to the total amount of structure in the system, one might imagine that the system could be prepared to recognize any particular environmental structure it was encountering. But the reverse is generally true: the amount of structure in the system is small compared to the amount of structure in the environment and is in no way capable of encoding it all, no matter how indirectly. Thus, a primary issue for the system is how it maps the external task into its internal structure.

We call this the assimilation issue, taking the term not only from its everyday use, but also because it corresponds well with the Piagetian use of the term, wherein a child assimilates aspects of its environment to the child's internal schemas (e.g., Piaget and Inhelder, 1969).

The status of assimilation mechanisms is rather peculiar in artificial intelligence. Every system must contain some mechanisms for assimilation in order to work on a population of tasks. Yet often the solution is to code within highly restrictive agreements between the structure of the task environment and the expectations of the internal program, thus not facing any assimilation issues (for an example see GPS, Ernst and Newell, 1969). On the other hand, the task of natural language input, which has received much attention, may be viewed comfortably as an assimilation issue. Similarly, the subfields of vision and speech are devoted to assimilation. Thus, one would think we know a lot about it. But the poverty of mechanisms that deal with analogy and metaphor, or that formulate problems in an environment so they can be tackled by an existing general method (such as the simplex method of linear programming), attest to the primitive state on assimilation.

The category of assimilation leads naturally to ACCOMMODATION, which is the modification of the internal structure of the system to be able to cope with wider range of tasks. The term is again Piagetian (Piaget and Inhelder, 1969). Both assimilation and accommodation involve a change of internal structure -- indeed a system requires internal structure change (i.e., memory) to behave in other than simple ways. But assimilation involves a transient modification that leaves the system with essentially the same bases for further assimilations, whereas accommodation is the general term for the acquisition of new assimilative capabilities. Though the distinction between the two cannot be made sharply in phenomenal or behavioral terms, it is usually extraordinarily clear when the structure of a system is made explicit.

DIRECTIONALITY refers to the problem of how a system organizes itself to do a particular task posed by the environment. A wide range of mechanisms are possible. There can be a single method which is automatically evoked -- indeed, which simply constitutes the structure of the program. Alternatively, there can be a collection of methods, which then requires some selective procedure. There can be an apparatus of goals with an explicit goal tree, or the directionality can be distributed in the structure of the program as a series of local decision processes. The system can rely on the recognition of progress after it has already been made, or it can do much processing so as never to try inappropriate courses of action.

A particular subissue is that most methods have only a limited chance of solving a task. Many systems simply give up once they have tried these methods and they have failed. We call the ***Keep-progressing issue*** the question of whether a system has an indefinite capability for continuing to work on a problem in a relevant way.

EFFICIENCY would seem to be almost self-explanatory. However, as we use the term it needs to be distinguished from directionality. In heuristic search paradigms a central problem is to control the combinatorial explosion. The heuristic devices that do this -- by providing effective ways to direct the search -- are part of directionality. But beyond such problems are also ones of computational efficiency, especially for high frequency basic processing tasks.

A clear example of efficiency arises with the use of interpreters. As is well known, the use of an interpretive language entails a cost of anywhere from a factor of two to ten over a compiled version of the same language. Analogous degradations occur in large systems when they are constructed with interpreter-like levels. What is purchased for this cost is flexibility and control. In practice the interpretive level can itself be handled interpretively, thus compounding the problem (but presumably also the gain). The ***Interpreter issue***, then, is whether a system has availed itself of these gains and if so what design features does it have for controlling the costs.

A second pervasive efficiency issue arises from the constraint of ***immediacy*** -- that the system is forced to exhibit its understanding within a limited time. More generally, we usually assess understanding to be the more complete, the faster it is exhibited. The efficiency issue here shows itself not only in how quickly basic processes can be performed, but in whether the methods used stand a chance within the time limits available. In general, search must be avoided, except in judicious amounts.

A final efficiency issue has been labeled that of ***formality***. In fact its point is somewhat speculative. The use of formalisms implies the adoption of fixed conventions for representation and processing, which provide guarantees of various sorts (that computations can always be carried out, that certain algorithms can be proved to give the correct results, etc.). Formalisms tend to the adoption of uniform procedures, which is to say, procedures that are applied unadaptively. Thus the introduction of formalism would seem to be a pervasive source of inefficiency in systems, leading to excessive processing over large domains for which the formalism yields large, complex expressions. There would appear to be a close relationship between this concern and that of multiple reprepresentation.

ERROR is ubiquitous. Starting with assimilation, which already implies an approximate relationship between the internal structure of the system and the task environment, errors are possible at every turn in a system. They arise from the methods themselves being only heuristic, from underlying unreliability of processes (which however, can be brought to very low levels in digital systems), to the fact that knowledge is missing on which to base solution so guesses must be made. Thus, all systems must have mechanisms to deal with error -- to detect it, to contain its effects, to correct it.

A special case of the error problem has come to be known as the ***frame problem*** (Raphael, 1970). In a modifiable environment, knowledge of the environment is not invariant under changes in the environment. This generates error, since once-true data is now false. Its name derives from the notion of a changing frame of reference.

Much of the notion of DEPTH OF UNDERSTANDING seems to be captured in being able to utilize knowledge whenever it is appropriate to do so. The question of task diversity discussed earlier responds to this issue: how to pose a wide enough variety of occasions of use to assess how deep is the understanding of some knowledge. The metaphorial adjective of depth (rather than say breadth) is perhaps a little misleading. Whether knowledge is appropriate and in what way, can vary in obscurity; thus deep understanding implies the ability to use the knowledge in nonobvious contexts. This connotation seems appropriate. But depth is also used when referring to the total knowledge of a topic (as in knowledge of quantum mechanics). Here it refers to the amount and completeness of such knowledge and the ability to answer successive "why" and "how" questions. This connotation is not appropriate, since we have not dealt with understanding of areas or events.

In systems with narrow task domains knowledge can be organized so as readily to be brought to bear on the task. However, as the diversity of tasks increase, there is no way to pre-organize the system so that the knowledge relevant to each task is already localized -- the subsets of knowledge demanded cross-cut each other in too many ways. Thus there arises the issue of ***access*** to the knowledge in a system, implying by this mostly the discovery of appropriate knowledge, rather than physical access. However, the two are related, since how information is represented has much to do with its access, especially when inferential processes are viewed as an access method (as when the access to 7 > 5 is made via stored knowledge of 7 > 6 and 6 > 5).

Having run through the list of Figure 2, we reiterate its function. It is a list of ***design issues*** -- of the key questions to be asked of any proposed understanding system as to how it works and what expectations might be generated about its viability. The list being functional cannot claim to be a unique decomposition, nor can it claim to be complete. It does cover the main issues we wish to raise in our understanding of understanding systems. It serves to perceive the existing state of such systems in artificial intelligence and to view Merlin as a potential contribution to that evolution.

SOME EXAMPLES

Let us put some flesh on this list of design issues by looking briefly at some existing understanding systems.

Predicate Calculus Theorem Provers

A useful start is the predicate calculus theorem provers -- they have simple structure and for many people they seem to be at the opposite pole from systems that understand. Figure 3 shows the layout of a prototypical resolution-style theorem prover. Figure 4 provides an outline description in terms of the design issues.[i]

The representation of knowledge in an RTP (Resolution Theorem Prover) is in terms of propositions, called clauses. Each clause is a disjunction of a set of predicates or negations of predicates, the predicates themselves being functions of variables over a domain of discourse. The entire knowledge in the system is encoded as a set of such clauses (taken to be the conjunction of the propositions), with the exception of the fixed structure (shown to the right in Figure 3) that processes the clauses in a task-independent way.

Resolution is a formulation of the first order predicate calculus and thus has the same representational scope. What can actually be represented depends on the particular predicates, but these are defined entirely within the system (by axiomatization in terms of the clauses in which they participate). On the other hand, a serious limitation is that the discourse refers to an external domain and does not permit representation of the strategies of processing or of the current state of the computation.

The representation also has some grain problems. An example shows up in the handling of equality. It can be axiomatized by three clauses corresponding to [X=X], [if X=Y then Y=X], and [if X=Y and Y=Z then X=Z]. However to effect the replacement of equals with equals, a separate clause is required for each variable position of each predicate, e.g., for P(X,Y) one must add two clauses: [P(X,Y) and X=Z implies P(Z,Y)] and [P(X,Y) and Y=Z implies P(X,Z)]. These extra clauses increase the combinatorial explosion, since they increase the size of the set of clauses.

A strong feature of resolution theorem provers is their uniform representation for all knowledge. Thus they opt to ignore the benefits of multiple representations. However, the matter is not so simple, for options still exist for how to encode a given task. The freedom lies in choice of predicates. For example, in representing a set of objects in space, one can choose a distance predicate D(X,Y) between objects or one can choose a pair of coordinate predicates, V(X) and H(X). As we know from analytic geometry, the same problem may look very different in the two cases. However, little choice exists for the style that axiomatization takes. In particular we cannot actually adopt a coordinate system and calculate directly within it. Neither can we develop

[i] For a straightforward account of resolution theorem provers in their own terms see Nilsson (1971).

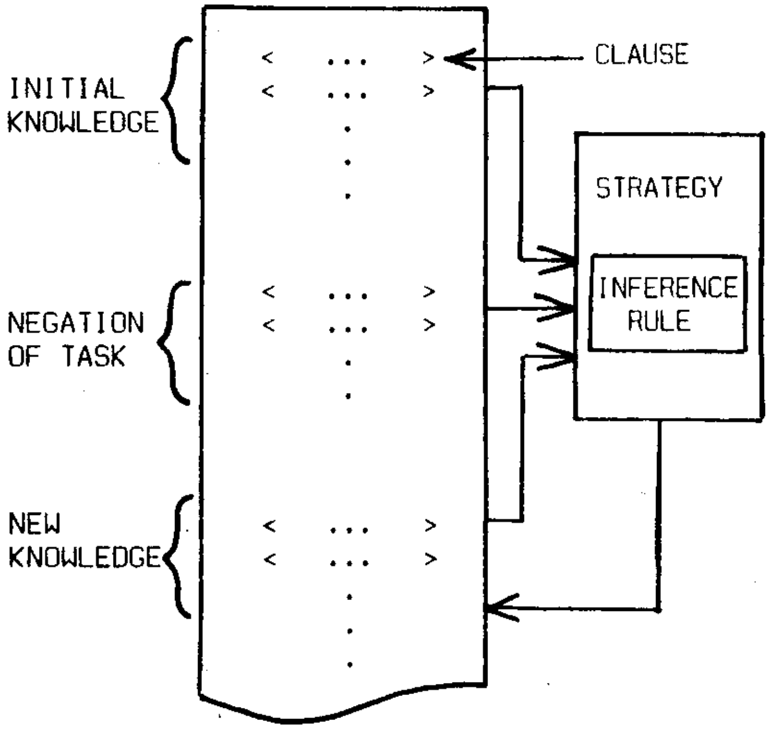


Figure 3: Diagram of Stylized Resolution Theorem Prover

REPRESENTATION	CLAUSES
SCOPE	COMPLETE (ONLY 1st ORDER)
GRAIN	IGNORE (E.G. EQUALITY)
MULTIPLE REPRESENTATION	IGNORE
ACTION	INFERENCE, EXTRACTION
ASSIMILATION	ADD CLAUSES, IGNORE ENCODING
ACCOMMODATION	(= ASSIMILATION)
DIRECTIONALITY	TEST FOR DONE, STRATEGY
KEEP-PROGRESSING	UNTIL FAIL/SUCCEED
EFFICIENCY	IGNORE
IMMEDIACY	IGNORE
FORMALITY	ACCEPT
ERROR	NO ERRORS
FRAME PROBLEM	NO CHANGES IN MODEL
DEPTH	COMPLETENESS
ACCESS	STRATEGY, LINEAR SCAN

Figure 4: Design Characterization of Resolution Theorem Provers

new representations or shift from one to another, since the set of predicates is fixed. (It would be possible to work with several representations simultaneously, though it has never been tried as far as we know.)

The action in resolution comes about through the application of a single rule of inference (called the Resolution Principle) plus the application of a strategy. The rule takes as input two clauses and produces a collection of new clauses (the resolvants) that can be added to the total pool (see Figure 3). The strategy determines which pair of clauses shall be considered next. Essentially all of the research in predicate calculus theorem provers has gone into the discovery of strategies and the establishment of their properties (including therein new rules of inference, which are ways of packaging selection rules). The great virtue of this arrangement is that all of the knowledge in the system (the clauses) can be processed in a uniform way to yield whatever new knowledge the system is capable of.

Assimilation and accommodation can be considered jointly, since an RTP makes no distinction between them. A new task is presented to the system by expressing it as a set of clauses (logically, the negation of the assertion of what is to be found or proved). Once added, these are indistinguishable from any other clauses in the system. Likewise since all knowledge is clauses, new knowledge (essentially accommodation) is acquired just by adding clauses to the system, either by an external source, or by permanent retention of clauses generated by the system itself. Thus assimilation and accommodation are both trivial operations (though the determination of what to assimilate or accommodate is not). One would expect, and indeed it is true, that a price is paid for this: The total knowledge structure is an unorganized mass. There are, of course, a number of theorem provers that have deviated from this homogeneous set of clauses. Insofar as they do have assimilation and accommodation, they require processing in order to integrate the new material into whatever organization has been set up. But the basic system, as described, occupies an extreme point with respect to these two design functions.

The above picture of assimilation is incomplete. An RTP is a totally internal system, with no provisions for acquiring problems in the external world and creating the internal representation as a set of clauses. It does not respond to this basic aspect of assimilation at all. In practice, of course, various solutions to this problem have been adopted, ranging from simple interfaces in mathematial notation, to English language interfaces (Coles, 1969), to interfaces that connect the RTP to an external physical environment for a robot (Nilsson, 1969). All of these represent essentially external solutions, so that the system itself cannot be said to face the task of assimilation.

Directionality is achieved in an RTP by a recognition that the answer has been found. The occurrence of a null clause signifies a contradiction (the occurrence of a clause C and another clause ¬ C) and the system remains continually sensitive for the production of such a clause. This is as minimal a scheme of direction as is possible, for in the basic system no shaping of the generation of new clauses occurs.

Again, some strategies do involve giving preferred status to resolutions that might lead toward the null clause (e.g., the Unit Preference rule) or to these using the clauses corresponding to the task (e.g., the Set of Support strategy). These convey a simple sort of directionality to the system, though they do not amount to a full scale goal system (e.g., that could set up subgoals).

In terms of special mechanisms, efficiency issues are ignored in RTPs, though of course many are coded in machine language to be as fast as possible. With the simplicity of the basic processing scheme, almost all efficiency issues translate into strategic issues of selection of appropriate clauses, which is to say, directionality. The system, of course, is the essence of a formalism. In fact all versions spend most of their time calculating according to the dictates of a particular uniform strategy and spend essentially no time considering either what to do or whether there are short cuts.

RTPs also ignore the problem of error. They assume the representation as given, that all acts of inference are valid, and that none are preferred to others because of reliability of the premises or whatever.

With respect to depth, however, the situation is different: the nature of the formalism guarantees that if there is a solution to the task within the knowledge that the system holds, then it will be found in due course. This is the essential content of the completeness theorems for the first order calculus (extended to cover the various selections introduced by the strategies). In terms of our present formulation, we must agree that an RTP understands its knowledge, since in fact it will use it whenever appropriate. Thus, RTPs receive something in return for the various uniformities that underlie them. Of course, this depth of understanding is guaranteed only if time bounds are ignored and in general the RTPs do not respond at all to the issue of immediacy.

This completes our general view of theorem provers according to the general design issues of Figure 2. It suppresses most of the detail of the strategies (wherein much of the research has resided), but reveals a number of gross features that dictate the sort of problems one will have in taking an RTP as an understanding system (as has been done in some of the work in robotics).

We can now introduce a theme that will be reiterated later. It seems to us that an understanding system must come to grips with the entire set of design issues. Thus, one of the functions of the list of Figure 2 is to make it apparent when a particular system is to be taken as only a partial proposal for an understanding system.

Planner-like Systems

An important development in the last few years is a collection of language systems called Planner-like systems after Hewitt's Planner (1972). Another one is QA4 (Rulifson, Derkson and Waldinger, 1972), and their features are migrating into many additional systems. Although they are languages, they incorporate a set of mechanisms that respond rather directly to the issues in our design list and any system erected within them adopts these solutions. They provide an appropriate contrast point to the RTPs just discussed and indeed some of their features were provoked by the system Zeitgeist provided by the work in theorem proving.

Figure 5 shows the design specifications for Planner-like systems.[i] The representations in Planner-like systems move away from a highly particular structure (i.e.,the clause, with a single semantic interpretation) to general data structures. However, they do not go so far as the list languages (such as Lisp) which adopt a single homogeneous data structure. Rather, in an effort to buy both ease of representation and efficiency, they provide a small set of basic data structures. Thus, QA4 provides lists, bags and sets (collections of symbols that are respectively: ordered with repetition of items, unordered with repetition, and unordered without repetition). Planner admits lists, strings and arrays, but not bags or sets. Thus, all structures are still to be composed out of some basic structures, but a small variety is admitted. In this respect they do not come fully to grips with the multiple representation issue, though there is still the freedom to represent a problem within these basic structures in any way that is desired. Furthermore, as we will discuss below, a strong pattern match capability is an integral part of these systems. This makes the encoding of representations easier, though there are yet no striking examples that show it.

REPRESENTATION	{LIST, BAG, SET, ARRAY}
SCOPE	UNRESTRICTED QUANTIFICATION
GRAIN	NO POSITION (BASE LANGUAGE)
MULTIPLE REPRESENTATION	PATTERN MATCH
ACTION	PROGRAMS (= LISTS)
ASSIMILATION	PATTERN MATCH
ACCOMMODATION	WRITE PROGRAMS
DIRECTIONALITY	GOALS + WEAK METHOD
KEEP-PROGRESSING	NO POSITION
EFFICIENCY	EQUIVALENCE
INTERPRETER	COMPILE IN BASE LANGUAGE
IMMEDIACY	NO POSITION
FORMALITY	ACCEPT BUT RICH
ERROR	BACKTRACK (+ MULTIPROCESS)
FRAME PROBLEM	UPDATE DATA BASE
DEPTH	
ACCESS	GLOBAL DATA BASE (PATTERN)

Figure 5: Design Characterization of Planner-like Systems

Planner-like systems are not inference systems built around a proof procedure. Rather they are language systems. Consequently the constraints that keep RTPs limited to first order do not apply, namely technical difficulties in developing matching

[i] We will operate from a composite of Planner and QA4, and ignore features of other systems, even where they augment these two.

(unification) for higher order logic (but see Huet, 1973). These systems then permit quantification over anything, i.e., over predicates and processes, and in general provide an omega-ordered predicate calculus (without thereby providing any validity-based proof procedures). Furthermore, in common with other list processing languages (of which they may be considered an extension) they permit self-reference to the programs and evolving data structures.

The Planner-like languages have no special proposals to make about grain and their position on this score is fundamentally that of the underlying list processing systems.

A basic tenet of these systems is that knowledge should be stored in the form of programs rather than in the form of propositions (the phrase used has been "procedural embedding of knowledge"). The position taken on action is to encode knowledge in a form that can be converted directly by an interpreter (of about the complexity of a list language interpreter). Consistent (again) with the embedding in an underlying list processing environment, programs are represented in lists, hence do not introduce a separate data structure.

A second basic aspect to the action is the concept of evoking the next procedure by means of a pattern match. The effect is to produce a form of generalized inference, in which structure X matches program [X → P] thus evoking (i.e., detaching) P. P itself may present a pattern Y which might match the program [Y → Q], thus evoking procedure Q. Hence chains of inferences (either forward or backward) can occur, intermixed with other processing. Since the processing of propositional representations is almost entirely governed by substitution and detachment in various guises,[i] there is no need to distinguish clearly between propositional and procedural forms.

We have now mentioned the pattern matching capability twice, once with respect to multiple representations and again with respect to the mechanization of an inference capability embedded within procedures. Pattern matching arises yet again with respect to assimilation. It is a step in the right direction. The pattern represents structure in the system that is tied to the knowledge about what to do (i.e., Pattern → Procedure). When a pattern successfully matches the external task structure, then that external structure is assimilated. Not only is the action evoked, but the variable parts of the external structure have been bound to the variables in the pattern (as when the pattern [X + Y] matches the expression [2 + 3] with the binding of X to 2 and Y to 3), thus becoming external structure that the system can deal with because the system has imposed an interpretation on it, namely that it fits the pattern.

Thus, to us, the Planner-like languages propose a specific mechanism for assimilation. All the Planner-like systems have very powerful match facilities, meaning by this that the class of patterns they admit is relatively large and complex. However,

[i] Indeed it is unclear whether there is any other kind of processing of symbolic structures except substitution and detachment, except for the processes that govern their evocation, storage and transmission (i.e., PMS level processes, Bell and Newell, 1971).

the pattern match is still a syntactic device in which the mappings that are admitted are those attainable with substitution operators.[i]

The added complexity that arises in a pattern-directed, procedurally-oriented system implies an issue of accommodation. Unlike the RTPs, which can add new structure simply be storing clauses in the data base, in Planner-like systems new capabilities imply new programs. The position taken by these systems is that they should engage in the construction of new programs. Some progress has been made on this though it cannot be said that the language systems themselves really provide other than an orientation.

Another major proposal of the Planner-like systems is the incorporation of a goal scheme as part of the basic language system, to provide direction. Thus, these are the first operational goal-oriented programming languages. There is an explicit goal structure that admits a pattern as the defining criterion of the desired object. In addition there is a generalized weak method.[ii] This takes the schematic form:

```
Is there a data structure that matches G?
    If so, quit with goal satisfied.
Is there a program of form [G → P]
    If so execute it.
    If not, fail.
```

The procedural expression [G → P] is to be read "If you wish to obtain G, then execute procedure P." P may be any procedure at all, including one that sets other goals. Thus via this generalized method one can evoke more specialized methods, including direct computational algorithms for obtaining the goal.

The Planner-like systems have not been strongly responsive to efficiency issues (except in one important particular, described under depth of understanding). However, one exception in QA4 also makes clear the nature of efficiency issues that do not come under directionality. Often one is dealing with classes of representations that are equivalent under common sets of operations. Naming (especially with bound variables) is one example, so are commutativity and associativity. The establishment of such equivalence often occupies a major amount of processing.[iii] Thus, in QA4

[i] There have been matches that do somewhat more than this. SCHATCHEN, the match in Moses' SIN (1967) matched relative to the commutative law and implicit factors ($X \leftrightarrow 1*X$, $X \leftrightarrow 0+X$), though as we shall see below this is approached in another way by QA4. GPS (Ernst and Newell, 1969) has a generalized set of so-called immediate operators in its match routine that are difference driven. Thus, GPS would match AvB to $\neg A \supset B$. The substitution operations were realized by means of these immediate operators. This scheme was also used for commutative and associative matching. See the generalized description of matching in Newell (1969).

[ii] See Newell (1969) for the concept of a weak method.

[iii] Those familiar with RTPs will note the amount of time spent in dealing with the axioms of associativity, which are complex. They introduce much of the thrashing that goes on in proving simple theorems in group theory.

expressions are kept in canonical form relative to the known properties of the operations involved. Much effort has been devoted in the system to making this canonicalization efficient.

The Planner-like systems also make a major proposal with respect to error: They incorporate backtracking as a basic control feature in the language. At user-specified places in a program (and automatically at certain places) the system prepares itself for a return in case of failure. It is prepared to undo all of the modifications made in the program after the backtrack point so as to re-establish the control and data context. Some such facility is required to make the language goal-oriented, since almost all problem solving methods introduce an appreciable element of search and with it the necessity to recover from unsuccessful attempts. Thus, the systems have at the language level the ability to do heuristic searches with essentially no further programming.

Backtracking, which is depth-first search, is well known to have serious drawbacks. In fact, the systems provide more complex control (e.g., multiprocessing) to help alleviate these difficulties. However, this aspect, which is the one that originally made these systems well known, has already caused the first major cycle of revision (Sussman and McDermott, 1972). Backtracking, though a general strategy for dealing with error and while a required capability, is simply too crude a tool and leads too easily to combinatorial explosion.

The final design issue is that of depth of understanding. Here again, Planner-like systems have proposed a mechanism, at least for the issue of access. Both Planner and QA4 have global data bases. All of the knowledge for a problem is put into this data base. Access is by pattern matching. (E.g., the matches for goal capability are taken with respect to the global data base.) The net effect of introducing such a memory system is that the usual design considerations of many specialized memories, with access privately known only to particular procedures, are set aside. All data in the system becomes available to all processes and will be accessed if it matches the patterns presented by a process as representing its needs.

A global data base poses a technical problem. Unless access to the base is essentially independent of total size, the system cannot afford to use it. For example, the RTPs achieve total access by two means: (1) using only strategies that are complete (that guarantee that no other clauses except those they will eventually select are necessary to solve the problem); and (2) serially scanning the set of clauses (i.e., the total data base). This later is a genuine barrier to large data bases. Thus, in the Planner-like system much attention has been given to the sophisticated indexing schemes that achieve the result of essential independence of the access time from the size of the total data base.

We have come to the end of our description of Planner-like systems in terms of our list of design issues. In several ways our description cuts across those usually given. The matching facility is normally described as a structural entity, whereas we have introduced it in the several places where it appeared to be the key element in meeting different functional requirements. Likewise, we have underemphasized the control issue, which in fact has operated as a major integrating focus for those working on the development of such systems (see Bobrow and Raphael, 1973). Our reason has been to relate the features of these systems to the main functions where we wish to understand what the systems have to offer. One aspect we should have stressed more was the total gain that comes from embedding all of these mechanisms in a language. Technically speaking, language embedding forces combinational freedom, so that ultimate patterns of behavior can be composed ***ad libitum*** from all the various components. From a design point of view this seems seldom to be achieved outside of the framework of language design.

It should be clear by now why Planner and QA4 have had the impact they have, for they contain at least five major ideas in the development of artificial intelligence systems:

(1) reversion to procedural embedding
(2) pattern directed evocation of processes
(3) goal-oriented language systems
(4) back-tracking
(5) global data bases.

Of these, the global data base and the pattern matching seem the most important. From a technical point of view in realizing them, all of these features (or their equivalents) are required concurrently. In this sense they are not independent. But they are independent in that they address functionally independent issues for the nature of understanding systems.

It should also be apparent why we treat these languages systems as proposals for understanding systems. If we considered some of the systems built within them (only Winograd's SHRDLU (1971) coded in Microplanner is well documented in the public domain), we would have to take all the discussion given here as starting point, though there would be some additional proposals as well.

Humans as Understanding Systems.

What do we know from psychology that would contribute to these design issues? That humans can in general understand and that they are so organized that they provide solutions of some sort to all the design issues goes without saying. The question is whether there are empirical data of sufficient clarity on any of the issues to indicate the nature of the solutions.

There appears to be little help. The difficulty seems to reside in the nature of the questions: they ask after fundamental mechanisms and structures, while the study of human behavior generally yields a different kind of data: behavioral regularities under controlled experimental conditions that can be used to test theories.

With respect to representation, for example, we know that the human is capable of symbolic representation, and that this is superimposed in some way on a system for representing the the world directly sensed through vision, hearing, smell, etc. We know that he uses a great range of multiple representations. We also know he has has much knowledge that is vague. But all these are external aspects and except for posing criteria, they do not indicate the nature of these representations. A similar story holds with respect to each of the other design issues. Though the terms assimilation and accomodation have arisen from Piagetian analysis of children's behavior, the theoretical notions behind them (e.g., schema) have little substance in terms of information processing structure.

A few things seem worth remarking. For instance, there is good evidence that humans do search -- that they backtrack in the face of errors. But such backtracking is quite limited and does not seem to be the major mechanism in response to error. Thus when errors turn up after substantial time periods (e.g., the revelation of the Piltdown man as a hoax), no attempt is made to backtrack -- indeed none of the memorial apparatus appears to be set up to backtrack. Some immediate consequences of the new information are found and corrected, but the large store of knowledge that might have some dependence on the now-erroneous knowledge is left to shift for itself.

There is also good evidence that the human gains access to an immense array of knowledge in very short times (relative to his time to do any processing at all). Thus, it would appear plausible that he has a global data base of some sort, and indeed this is assumed in most psychological theories (so-called LTM). That he cannot recollect everything in the data base immediately (as has often been shown -- e.g., what was the name of your fourth grade teacher?) is immaterial, because the momentary retrieval properties of LTM must depend on the presented address (i.e., retrieval cues) as well as on the contents of the memory.

Finally, some case can be made for excluding various possibilities for the human action mechanism -- i.e., for the way whereby the human obtains action from his knowledge. There is no evidence that all action exists as plans which are interpreted by a process; humans seem often to be able to react directly. Likewise action does not all derive from an inference technique, which then flows into behavior by a final decoding; much of his knowledge seems to be held in action form. Humans of course are capable both of following plans and of making inferences. The issue is what knowledge-action conversions mechanisms form the basis of these systems. (E.g., see Newell and Simon, 1972, for an account of one such proposal, production systems.)

MERLIN

With these design issues as background we turn to Merlin. To reiterate: Merlin has been an exploration into the design of an understanding system. It has existed in several operational versions, though none of them has been complete as far as the set of design issues is concerned. Thus, within the philosophy presented in this paper, each version is an exploration into selected aspects of the design. Nevertheless, many aspects of Merlin have become relatively constant and represent distinct proposals in terms of certain of these design issues. We will present a composite picture of the system through Summer 1973.

The design issues will serve as a guide in the exposition. We present a summary of them in Figure 6, comparable to our summary for the other systems.

REPRESENTATION	MAPPED OBJECTS
SCOPE	UNRESTRICTED QUANTIFICATION
GRAIN	SELF-DEFINING COMPONENTS
MULTIPLE REPRESENTATION	NO POSITION
ACTION	PROGRAM (POTENTIALLY) AT EACH NODE
ASSIMILATION	FIND/CREATE MAP
ACCOMMODATION	NO POSITION
DIRECTIONALITY	GOALS = MAPPINGS
KEEP-PROGRESSING	TRY OTHER ALTERNATE VIEWS
EFFICIENCY	
INTERPRETER	COMPILING = SIDE EFFECT OF REFERENCE
IMMEDIACY	NO POSITION
FORMALITY	MINIMAL
ERROR	NO POSITION
DEPTH	NO POSITION

Figure 6: Design Characterization of Merlin

Representation

Merlin has a basic data structure called a *β-structure.* Its general form is:

$$\alpha : [\beta\ \alpha 1\ \alpha 2\ \ldots] \qquad (3)$$

This may be read: "α is a β further specified by $\alpha 1$, $\alpha 2$, ..." The name "β- structure" derives simply from the practice of writing the structure with the particular notation above. However, it serves to keep the interpretation of the structure somewhat free of preconceptions.

The intended interpretation can be approached via a variant reading: "α can be viewed as a β given that $\alpha 1$, $\alpha 2$ ... .". Its source rests in the emphasis we have already given to the issue of assimilation. A fundamental problem for an understanding system (indeed for any information processing system) is making contact between what it actually knows (its structure) and what it does not yet know. Thus, in (3) β is taken as known and α is to be understood by being assimilated to β, which requires that further things be also taken as given (the $\alpha 1$, $\alpha 2$ ...). We call β the ***schema*** and the $\alpha 1$, $\alpha 2$ ... the ***components*** of the β-structure. (By way of notation, the schema of α is written: $\alpha\uparrow$; the set of components of α is denoted: α#.) Again, the terms are intended to be relatively neutral, since the actual interpretation to be given the β-structure depends mostly on the processes that manipulate and interpret it.

The components are themselves simply other β-structures, that is, other specifications that something is to be viewed as something else along with what it takes to do so. Thus, the β-structure gives a structure in terms of which some item is to be understood, along with the mapping that is required so that the object may be assimilated.

β-structures serve the corresponding role in Merlin to nodes in a semantic net memory or to clauses in a theorem prover. All knowledge is to be coded in their terms. We will often refer to β-structures as nodes and to the set of all β-structures as the ***Knowledge Net*** (KN). However, β-structures are not process specifications; there is also a general representation for processing (called the action), which is dealt with later.

Let us consider some simple examples.

SAM: [BOY [EYES BLUE]] (4)

SAM is a BOY further specified to have EYES that are further specified to be BLUE. BOY, EYES and BLUE are of course themselves β-structures, which is to say, are understood because they can be viewed in terms of yet other things, themselves understood. (This infinite regress will be terminated shortly.)

```
LT: [HEURISTIC-SEARCHER [TASK LOGIC]                          (5)
                        [AUTHOR NEWELL-SHAW-SIMON]]
```

LT (an early theorem prover) can be viewed as a HEURISTIC-SEARCHER whose TASK is further specified to be LOGIC and whose AUTHOR is Newell, Shaw and Simon. In these examples we leave out most of the detail that would be required for an adequate description. For an example of a full-fledged description of LT (in Merlin 1) see Moore (1971).

In (4) and (5) the relationship of node to schema appears to be the IS-A relationship, familiar from many semantic nets. But the interpretation is to be taken more broadly.

8: [+ 3 5] (6)

Thus, 8 may be viewed as a + further specified by 3 and 5. Here a β-structure appears as an expression comprised of a function (+) with its arguments. For this interpretation to hold, the operation + must be viewed as a generalized element of its range; i.e., + can be viewed as an indeterminate number. Given this, we can have the following:

X: [+ 3] (7)
Y: [X 5] (8)
Z: [+ 3 [> 5]] (9)

Here X is a + further specified by 3, which is to say either the function of adding 3, or an indeterminate number that is 3 more than another unspecified number, depending on how you wish to view it. Then Y is X further specified by 5, which is to say, it may be viewed as 8. Similarly, Z is another indeterminate number greater than 5. In these we get the typical heirarchy of functional composition.

A β-structure may have (simultaneously) a number of ***alternate views***. Thus, a β-structure called P35 might be viewed in the following ways:

```
[+ 3 5]                                                    (10)
[8]                                                        (11)
[ARITH-OP BINARY COMMUTATIVE [FUNCTION ADDITION]           (12)
        [FIRST-ARG 3][SECOND-ARG 5]          ]
[WFF [MAIN-CONNECTIVE +][LEFT-HALF 3][RIGHT-HALF 5]]       (13)
...
```

At any given point in time, exactly one of these is the ***principal view***. Whenever we write, e.g.,

P35: [+ 3 5] (14)

we indicate that [+ 3 5] is the principal view.

Merlin generally deals with a β-structure only in terms of its principal view. The others are kept as alternatives, and are referenced only by an explicit attempt to change the way the object is viewed. An entity gets a new principal view as the result of a successful mapping of the structure into the new view.

If Merlin finds a map from one β-structure (Y) to another (X), then it has found a way to view X as a further specification of Y. The new view of X will have Y as its schema and its components will be the various aspects of the map from Y to X. This leads to yet another reading of the β-structure: (3) may be read as "α is the result of applying to β the mapping represented in the α1, α2 ... ." We indicate mapping Y into X by the notation: X/Y (to be read "view X as a Y" or "map Y to X"). This attempt may succeed or fail, depending on the relations between X and Y, and how hard Merlin is currently willing to try. If X/Y succeeds, a map has been found from Y to X; suppose this map could be represented by the set of β-structures: X1, X2, ... , then the result of the successful map is to assign X a new principal view:

X: [Y X1 X2 ...] (15)

Let's look at some examples. Suppose we know:

MAN: [MAMMAL NOSE:[...] HOME:[...]] (16)
PIG: [MAMMAL SNOUT:[...] STY:[...]] (17)

and we pose Merlin the task of viewing a PIG as a MAN (i.e. PIG/MAN). Assuming that the indicated sub-maps succeed, the result would be:

PIG: [MAN SNOUT/NOSE STY/HOME] (18)

Equivalently: "A PIG can be viewed as a MAN if his SNOUT is viewed as a NOSE and his STY is viewed as a HOME." On the other hand, if we assume that a PIG's TAIL cannot be viewed as his HOME (i.e. TAIL/HOME fails), and if we had, instead:

PIG: [MAMMAL SNOUT:[...] TAIL:[...]] (19)

then PIG/MAN would also fail.

The β-structure representation (considering only the principal views) induces a classical hierarchy of knowledge. This is illustrated in Figure 7, which shows (albiet schematially) a hierarchy of knowledge for LT. The topmost entity is PROCESS, which could of course be defined in still more rarefied terms, e.g., as a SYSTEM with certain properties, a SYSTEM itself being defined as ... etc. A problem-solver (PS) is viewed as a kind of PROCESS, a heuristic-searcher (HS) as a kind of PS, LT as a kind of HS, LT with a set of axioms given (called LTA) as a kind of LT and LT with both axioms and a theorem given (called LTAT) as a kind of LTA. Shown to the side, even more briefly, is hill-climber (HC), considered also as a kind of PS, and GPS, considered as another kind of HS. Thus a full hierarchial tree would exist in the Knowledge Net.

```
                         PROCESS: [...]
                    PS: [PROCESS ...]
               HS: [PS ...]          HC: [PS ...]
          LT: [HS ...]          GPS: [HS ...]
     LTA: [LT   [AXIOMS AX1 AX2 AX3]]
LTAT: [LTA   [THEOREM T1]]
```

Figure 7: Hierarchy of Knowledge

Each node in the Knowledge Net is to represent something external to the memory structure. At any moment, one view holds a preferred status (as principal view). However, it may be dethroned at any time by another successful map. There is no view that represents, so to speak, what the entity, ***really*** is. In particular, the original view does not have such a status. We can capture this representational decision as:

> ***The system treats all its views of reality as contigent alternatives.***

There is an issue of representational grain here. Adopting this principle avoids having to make the decision to confer special status on some knowledge (that is the "true" view), which decision would be forced in order to get the knowledge represented at all.

There are three subissues under that of representation: scope, grain and multiplicity. There is no way to ascertain the scope from the structure, i.e., what sorts of operations can be performed and with what interpretations. However, it turns out that the system is unstratified (i.e., without syntactic restrictions), which permits manipulation and quantification over all the structures in the language, hence inducing omega-ordered completeness in the same sense as for the Planner-like language (namely completeness of representation but without an associated inference scheme).

More important in terms of the design is the issue of grain. One of the original design decisions was that components in a sense declared themselves:

> ***The interpretation of a component is to be based entirely on the contents of the component, the schema and the collection of mappings currently being attempted.***

There are no syntactic rules that dictate how a component is to be interpreted or what role it is to play. Each component presents itself as an expression for specifying further the schema with which it is associated. What that further specification is must be discovered by an attempt to interpret that component in conjunction with the others, given the context in which the entire β-structure occurs.

There are several consequences from this decision. The one of note here is the attempt to avoid the grain problem by refusing to set a fixed grain (formal rules of interpretation) on the components. The hope (though not yet well tested) is to be able to shade the components in whatever ways are appropriate to capture whatever indefinite state of knowledge the system needs to represent. Thus, a new β-structure is formable simply by taking a schema and combining it with whatever β-structures have been found or manufactured that seem to represent current relevant knowledge. These can be thrown together into a new β-structure and the new structure used instead of the old one. No conditions have to be met in order to form the new β-structure; hence there is one less situation where pre-commitment must be made and uncertainty absorbed. It goes without saying that this one design decision does not solve the grain problem, it only removes an impediment.

The ancient issue of the ontological status of individuals and concepts (particulars and universals) casts its shadow across the design of a representation: Shall the system make a fundamental distinction between individuals and generic concepts?[i] Merlin does not.

[i] See Anderson and Bower (1973) for a system that does make such a distinction sharply.

> ***The memory system has only a single representational scheme (the β-structure) to refer to both individuals and generic concepts.***

In particular, any entity may be further specified and the hierarchy shown in Figure 7 is indefinitely extendable downward, as well as upward. The system does not have to make a decision whether LT is an individual or only a concept (there being many versions of LT, differing from time to time, not only with respect to axioms and theorems, but with respect to the heuristics used, trace processes added, etc.). Merlin thus avoids another problem of grain. Such a design feature does not prevent Merlin from having a concept (UNIQUE) that serves to designate individuals, e.g., SAM: [MAN UNIQUE].

Merlin currently provides nothing except intent with respect to multiple representations. The philosophy behind β-structures extends directly to an entity having non-β-structure representation. However, with only a couple of minor exceptions (simple linguistic expressions and sequences of behavioral events) no way has yet been provided to re-view an external representation as a β-structure or vice-versa.

Let us summarize the representation; A Knowledge Net consists of a set of nodes (or entities), each of which has associated with it a collection of views (β-structures). A view, which forms the basic data structure, specifies another entity (the schema) in terms of which the given entity can be viewed, plus the set of further specifications (the components) required to make the mapping. The components themselves are entities (hence with β-structures). Nothing further is assumed in the structure of the Knowledge Net about how to interpret the entities. In particular the memory is homogeneous with respect to the notion of individuals, hence there are no atoms.

Action

The Knowledge Net is purely a static structure and the problem of generating action from structure exists for Merlin, as it does for all other systems. There are three aspects to the issue of action: (1) the implied logic of β-structures which permits various conclusions to be extracted from them; (2) the programs (called actions) associated with nodes; and (3) the integration of these aspects of action into a coordinated whole.

Mapping. The basic operation with β-structures is to ask for one entity to be viewed as another, i.e., for two given entities, X and Y, to find if X/Y has a solution. Before considering what is involved in this process in general, let us consider some examples. If we reconsider expression (4) and ask for SAM/BOY the attempt is clearly successful, since the SAM is already expressed as a further specification of BOY. If we added the expression:

BOY: [MAN YOUNG] (20)

and ask for SAM/MAN, then we could apparently again succeed.

SAM: [MAN YOUNG [EYES BLUE]] (21)

(21) is an adequate representation of SAM as a further specification of MAN, given the adequacy of (4) and (20). Clearly the relationship is transitive and we could expect to be able to develop X/Y if Y were anywhere up the schema hierarchy of X.

However, the actual β-structure is not necessarily determined by collapsing all the intermediate levels in the β-structure hierarchy (a process we call ***flatting***) and taking the union of all the components. Consider:

MAN: [PERSON EYES] (22)

If we now ask for SAM/PERSON there are two possibilities:

SAM: [PERSON YOUNG EYES [EYES BLUE]] (23)
SAM: [PERSON YOUNG [EYES BLUE]] (24)

The latter is evidently the one that is wanted. For [EYES BLUE] is a further specification of a component of MAN and this was explicitly noted by writing the schema EYES. When the flatting takes place the connection should be made in developing the correct β-structure. Suppose, instead of (4) SAM was defined by:

SAM: [BOY BLUE] (25)

Then in asking for SAM/PERSON we obtain the two possibilities:

SAM: [PERSON YOUNG EYES BLUE] (26)
SAM: [PERSON YOUNG [EYES BLUE]] (27)

Which is appropriate is now ambiguous, at least if it is considered possible that SAM himself is colored BLUE. Indeed, the only reason the additional alternative:

SAM: [PERSON [YOUNG BLUE] EYES] (28)

is not considered is that BLUE does not further specify a component of YOUNG. For this latter to be discovered, of course, requires an investigation of the representations of both BLUE and YOUNG.

Consider now the following simple versions of LT, a heuristic-searcher (HS) and a theorem-prover (TP) in terms of a problem-solver (PS):

```
LT: [HS [TASK [LOGIC WHITEHEAD-RUSSELL]]                  (29)
              [AUTHOR NEWELL-SHAW-SIMON]]
HS: [PS SEARCH]                                           (30)
TP: [PS [TASK LOGIC]]                                     (31)
```

Can LT be viewed as a theorem prover, i.e., can we find LT/TP? The solution is:

```
LT: [TP SEARCH [LOGIC WHITEHEAD-RUSSELL]                  (32)
                 [AUTHOR NEWELL-SHAW-SIMON]]
```

This result is arrived at by first expressing both LT and TP in common terms, namely viewing them as PSs:

LT/PS → LT: [PS SEARCH [TASK [LOGIC WHITEHEAD-RUSSELL]] (33)
[AUTHOR NEWELL-SHAW-SIMON]]
TP/PS → TP: [PS [TASK LOGIC]] (34)

If every component of TP has a further specification among the components of LT (is *covered*), LT can be expressed as a further specification of TP. In this case, [TASK [LOGIC WHITEHEAD-RUSSELL]] covers [TASK LOGIC], providing the solution.

The above examples have exhibited the form of the logic built into β-structure representation. The general case to be easily outlined:

```
Flat and Cover Procedure:

        Let X: [U X1 X2 ...] and Y: [V Y1 Y2 ...]

        To form X/Y:
                Find a common higher schema (call it Z)
                   If it does not exist, fail.

                Flat X to Z →  X: [Z X1' X2' ...]

                Flat Y to Z →  Y: [Z Y1' Y2' ...]

                Find a cover from the components of X for those of Y:
                   Find for each Yi' a distinct Xj' such that Xj'/Yi'
                      If cannot be done, fail

        The solution is:

                X: [Y X1"/Y1' X2"/Y2' ... Xn" Xn+1" ...]                    (35)
```

where the Xi" are some reordering of the Xi as dictated by the selection for the cover. The solution need not exist, as is evident. Neither is it necessarily unique. Each flatting operation gives rise to possible optional groupings and there may be additional ways to find the cover (which itself can have options for how the submapping goes). Whether there is actual ambiguity or whether these multiple options can all be resolved to yield a unique solution, depends on the content of the particular β-structures.

Although presented in a somewhat unfamiliar guise, the basic logic of the slash operation (/) is a reflection of the composition of mappings.[i] It does provide a basic

[i] If we consider entities to be sets of aspects (leaving open what constitutes an aspect in a given situation), then we can define mappings between entities as correspondances between their aspects, i.e., M:{aspects of entity 1} → {aspects of

capability whereby inferences of an elementary sort can be made given just the representation of entities in terms of β-structures, without further processing structure.

In some sense X/Y can be taken as a syntactic operation, defined on uninterpreted β-structures. We will often refer to it this way. Yet we may also take the β-structure as equivalent to a semantic definition (as that term would be used in a semantic memory). One way to see this is to reinterpret the β-structure as a collection of relational linkages to other nodes in the total network. Each of the components gives the relationship to another node. An example is shown in Figure 8 which makes this correspondence clear. The component of course can be more complex than shown, both in terms of what relationship is designated and the number of nodes (the remaining subcomponents of the component) to which the linkage leads.

The Flat and Cover Procedure does not exhaust the means whereby one entity can be viewed as another. A solution may be *posited* in order to consider the nature of other mappings made relative to that posit. For example, suppose we are looking for a new house, and have a description of what we are looking for:

```
HOUSE-WANTED: [HOUSE LRW:[LIVING-ROOM] KW:[KITCHEN]                (36)
                     BW:[BATH] BRW1:[BEDROOM]
                     BRW2:[BEDROOM] BRW3:[BEDROOM]]
```

Suppose we encounter:

```
HOUSE-FOUND: [HOUSE LRF:[LIVING-ROOM ... ] KF:[KITCHEN ... ]       (37)
                    BRF1:[BEDROOM ... ] BRF2:[BEDROOM ... ]
                    BF:  [BATH ... ]    DF:  [DEN ... ]      ]
```

Can this house be seen as a further specification of the one wanted? That is, does

```
HOUSE-FOUND/HOUSE-WANTED                                           (38)
```

succeed? Evidently, this mapping can succeed only if some way is found to cover the third BEDROOM (BRW3). We observe that the DEN (DF) is not needed to satisfy any of the sub-maps. If the attempt DF/BRW3 succeeds by posit (i.e., simply assume that DF can be viewed as BRW3), then the entire map will succeed. We indicate this by simply leaving DF/BRW3 as the final expression:

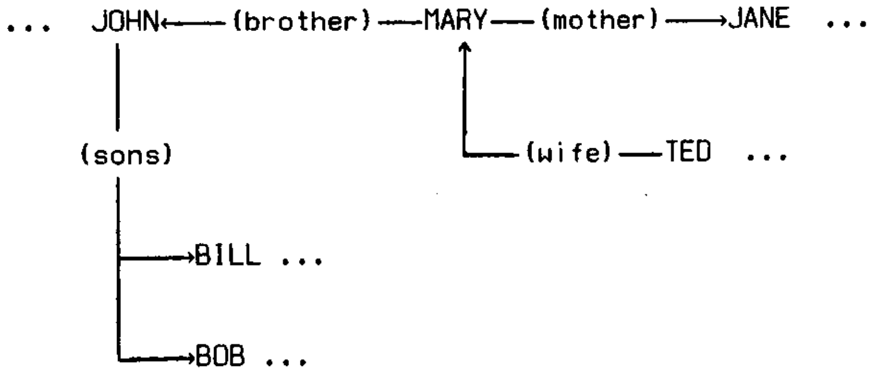
```
HOUSE-FOUND: [HOUSE-WANTED DF/BRW3]                                (39)
```

entity 2}. In these terms X/Y implies finding a mapping M: Y→X. If we have X/Y and Y/Z (i.e., X:[Y ...] and Y:[Z ...]) then X/Z can be found by taking the composition of the two mappings. The logic of flatting agrees with the associativity implied by these mappings. Likewise, finding a cover is the generation of a mapping by enumerating individually corresponding aspects. Failure to find a cover with a particular decomposition into aspects (the components of the flatted entities) does not preclude finding a map with a different decomposition.

The knowledge expressed in the following English sentences:

John is Mary's brother.
Mary's mother is Jane.
Ted's wife is Mary.
John's sons are Bill and Bob.

could be represented by the following semantic net:



In β-structures, this might look like:

```
JOHN:  [PERSON [BROTHER-OF MARY] [SONS BILL BOB]]
JANE:  [PERSON [MOTHER-OF MARY]]
MARY:  [PERSON [WIFE-OF TED]]
```

Figure 8: A Semantic Net Fragment Expressed in Terms of β-structures

Positing is a universal technique (which must therefore be used sparingly). In the opposite direction there can be other bases for forming alternative views of entities that are more specific than flatting. They all involve, in one way or another, the actions, to which we now turn.

Actions. The basic mechanism for obtaining behavior from knowledge is the ***action.*** To each β-structure is potentially associated an action, with the following interpretation:

> ***The action of a β-structure is its fully specialized process equivalent.***

What does it mean for a static object to have a "process equivalent"? All of Merlin's intellectual activity derives from the mapping operation. The purpose of the actions is to function as highly specialized versions of this operation. The action for X, A(X), embodies all knowledge localized at the β-structure in the form of a streamlined version of the Flat and Cover procedure. Thus, A(X) is capable of attempting any map to or from X (X/Y or Z/X). It would succeed under the same conditions as would the full-scale mapping process, and win or lose, costs at least an order of magnitude less computation (since at least one interpretive level has been circumvented).

To get an idea of how actions behave, suppose we want to view P35:[+ 3 5] as a NUMBER. We try:

P35/[NUMBER] (40)

A(P35) is capable of representing P35 as any of:

[+ ...]
[NUMBER ...]
[SUMMATION ...]
[- ...]
[EXPRESSION ...]

A(P35) has immediate access to 3 and 5 (with no searching, testing, etc.) as well as the full knowledge of +. When faced with (40), A(P35) discovers that what is desired is to re-view P35 as [NUMBER ...], which is one of its specialties, and it produces 8:[NUMBER ...] as the result of the successful match.

As an example of an action functioning on the other side of a map, consider:

M73/PNN (41)

where:

M73: [- M7:[7 ARG1] M3:[3 ARG2]]
PNN: [+ N1:[NUMBER] N2:[NUMBER]]

Suppose M73 has no action but PNN does. The first thing A(PNN) does is attempt

M73/+. This might succeed (e.g., by virtue of A(-)) producing: [+ M7 [NEG M3]]. A(PNN) then replaces the original problem with:

[+ M7 [NEG M3]]/PNN (42)

A(PNN) then searches in [+ M7 [NEG M3]] for any objects that can cover N1. M7/N1 and [NEG M3]/N1 both succeed. Since A(PNN) embodies the knowledge that + is commutative and that N1 and N2 are equivalent structures, it need conduct no search for any objects to cover N2. A(PNN) also need not concern itself with the possible (apparent) ambiguity between pairing M7 with N1 and [NEG M3] with N2, or vice versa.

From these examples we see that the mapping operation is in some respects strongly similar to matching, but represents an even more powerful process. The terminal operations in a match are limited (symbol identity test, variable assignment) whereas Merlin's mapping allows full semantics-driven re-representation, and even a few judicious posits, when necessary. Thus, for example, although P35 cannot be matched to [NUMBER], it can be so mapped.

Replacement of matching with a general mapping process has many consequences. One that requires mention, even though we will not treat it in detail, is the loss of control that arises from the failure of matching. Since a map can always be created between X and Y if sufficiently bizarre maps are allowed, no information is provided by obtaining X/Y. (Examination of AI programs will show that immense amounts of control are communicated by the success or failure of the match processes.) In fact, maps are not always found in Merlin, but control processes must exist to limit the range of mappings considered. In Merlin 2.5 these were buried in a fixed set of search processes; Merlin 3 will have a general scheme of controllers, but the design is not yet firm.

Integration. The two parts so far discussed provide the pieces out of which Merlin can compose integrated sequences of action. The basic issue belongs to directionality, but an important aspect can be discussed here:

> ***All behavior derives from the actions distributed throughout the Knowledge Net.***

In particular, the various attempts at mapping are performed by actions, providing their principal means for dealing with the Knowledge Net. There is no intelligent interpreter, i.e., a single active program that manipulates the Knowledge Net. There is an interpreter in Merlin, but its functions are limited to finding actions and to creating the nested context of β-structures within which the action is then executed.

The existence of actions on some subset of nodes in the Knowledge Net terminates the potentially infinite regress in interpretation and exploration. When, in attempting a mapping, one runs into an action, there is no need to go beyond that entity into its internal β-structures.

Assimilation

The main points about Merlin and assimilation have already been presented, since they lie at the foundation of its representation. Merlin puts the issue of assimilation at the center, not only of Merlin vis-a-vis the external task environment, but of each piece of knowledge in Merlin vis-a-vis other pieces of knowledge. One novely of this, perhaps, is the possibility of constructing an entire static knowledge structure out of the mappings of one structure into another. It is of course not so strange when re-viewed as a semantic-net-like structure, expression as in Figure 8.

Setting up the representation as the natural encoding of assimilation does not, of course, solve the problem of making assimilations. However, it goes part way. Consider a possible β-structure for logic viewed as a formal mathematical system:

```
LOGIC:[MATH-SYSTEM                                                    (43)
          AXIOM:    [WFF MULTIPLE]
          THEOREM: [WFF DESIRED]
          RULE-OF-INFERENCE: [RULE MULTIPLE
                                [RANGE WFF]
                                [DOMAIN WFF] ] ]
```

Next we propose a β-structure for a problem space, which is a construct belonging to the notion of a problem solver:

```
                                                                      (44)
PROBLEM-SPACE: [SYSTEM
                   INITIAL-OBJECT: [OBJECT]
                   FINAL-OBJECT:   [OBJECT DESIRED]
                   OPERATOR: [RULE MULTIPLE
                                [RANGE OBJECT]
                                [DOMAIN OBJECT] ]  ]
```

If a problem solver were to work on logic, then one requirement is that it would have to view logic as a problem space. In (45) we see the mapping as it comes about when simply the basic Flat and Cover Procedure algorithm is applied along with one posit.

```
                                                                      (45)
LOGIC/PROBLEM-SPACE → [PROBLEM-SPACE
                          WFF/OBJECT
                          AXIOM:[INITAL-OBJECT WFF/OBJECT]
                          THEOREM:[FINAL-OBJECT WFF/OBJECT]
                          RULE-OF-INFERENCE:[OPERATOR
                                                WFF/OBJECT]]
```

The posit is necessary if the two entities (WFF and OBJECT) are defined without common intersection (as well they might be). If we assume that WFF and OBJECT have been defined with little or nothing in common, then we would expect WFF/OBJECT to fail (unless posited). However, from the definitions in (43) and (44), we see that without this correspondance, there is little hope of finding any mapping from PROBLEM-SPACE to LOGIC. The mapping is first attempted with positing inhibited, and, as expected, it fails. However, in failing, Merlin has observed the number of places in which WFF/OBJECT was requested (and failed). With this as a clue, Merlin posits WFF/OBJECT and retries the mapping with other posits still inhibited. This time, the mapping works, producing the results in (45). This happens, of course, because of the similar internal structure of RULE-OF-INFERENCE and OPERATOR. If this correspondence were less transparent the mapping would have to drive further up the hierarchy, perhaps with additional posits. On the other hand if WFF and OBJECT had a common schema, say STRUCTURE, it might be possible to establish WFF/OBJECT without a posit.

In this example, and in several others to come, we exhibit a successful mapping in the pursuit of some particular functional aim. Here it is assimilation in its most classical terms: the mapping of a method onto a task environment, to be able to apply the method. The adequacy of the mapping techiques to significant problems cannot be concluded from these simple examples. We will discuss the issue later under the heading of natural intelligence. Our purpose in these examples is to exhibit the design decisions and to give some evidence of their plausibility, not to imply solutions to indefinitely difficult problems.

Accommodation

Merlin's thoroughgoing adoption of assimilation appears to generate difficulties with respect to accommodation. Figure 6 records NO POSITION opposite Accommodation, since we do not pretend to understand this issue yet. If Merlin copes with any new information by constructing a β-structure for it in terms of what it already knows, and processing it through that already-known schema, then how is Merlin ever to obtain anything new? It does obtain new β-structures and can use these as further schemas for yet other to-be-assimilated ojects. But these newly obtained structures derive their potency, so to speak, from their schemas, so that nothing really new is added. Following this path would seem to bring Merlin to mapping the whole world onto the set of primeval β-structures (they need hardly be primitive) with which it is endowed.

Possibly the gradual accretion of layers of further specification from a conceptually spanning set of nodes will suffice. More likely, ways must be found for environmental structure to construct new β-structures in the system, thereby adding new schemas to be used in future assimilations. A hint of how this might occur can be found in programming. Consider a β-structure for PROGRAM as a SEQUENCE of INSTRUCTIONS in some LANGUAGE:

PROGRAM: [SEQUENCE [INSTRUCTION MULTIPLE] [LANGUAGE]] (46)

with ADDXYZ as a particular further specification:

```
ADDXYZ: [PROGRAM                                        (47)
            [1ST [CLA VBLX]]
            [2ND [ADD VBLY]]
            [3RD [STO VBLZ]]
            [4TH [RETURN]]
            MACHINE-CODE      ]
```

The important feature is that while further specifications of this structure are still programs, their variety encompasses arbitrary behavior (in the way typical of all programs). Thus, what from one view is always a sequence of instructions can become from another view a method for achieving a goal. Indeed, if we now bring up a structure for solving a particular problem, expressed at a high level so that it could not be said the system actually knew how to achieve the goal, only that it wanted a way to do so, the program could be re-viewed as such a method. Furthermore, further specifications of the re-viewing will remain organized as methods for solving the goal, not just as a sequence of instructions.

This example may be used to illustrate another aspect of mapping. If one were to pose to a human (programmer) the task of expressing ADDXYZ as "an arithmetic expression involving the sum of two numbers," we would not be surprised were he to respond with something like: "VBLX plus VBLY." How might Merlin do this?

It would not be unreasonable for ADDXYZ to have, as an alternate view:

```
[ASSIGNMENT [← VBLZ] [+ VBLX VBLY]]                     (48)
```

This might be the result of a "decompile," i.e.,

```
ADDXYZ/[STATEMENT ALGOL]                                (49)
```

Alternatively, the view in (47) might itself have been the result of (48) being "compiled":

```
ADDXYZ/[PROGRAM MACHINE-CODE]                           (50)
```

in either case, given that ADDXYZ has (48) as an alternate view, and given that some variant of ALGOL permits statements of the form:

$$W \leftarrow (Z \leftarrow X + Y) \qquad (51)$$

(I.e., the language permits an assignment statement to be viewed as the value being assigned.) It is reasonable to suppose that:

```
ADDXYZ/[ARITHMETIC-EXPRESSION ALGOL]                    (52)
```

would yield:

```
[ARITHMETIC-EXPRESSION                                      (53)
      [CONNECTIVE +]
      [LEFT-HALF VBLX]
      [RIGHT-HALF VBLY] ]
```

as yet another alternative for ADDXYZ. With this as prologue, were Merlin faced with:

ADDXYZ/PNN: [+ N1:[NUMBER] N2:[NUMBER]] (54)

the reasonable conclusion would be:

ADDXYZ: [PNN VBLX/N1 VBLY/N2] (55)

Which is approximately how we speculated our programmer would respond.

This does not solve the accommodation problem. It does suggest how a process that selects instructions according to the task environment can add to a β-structure such that the final result is not intuitively to be described as just a further specification of some internal schema. The burden in this solution is on the task of programming, but at least one can see how the environment might select the instructions (e.g., by local searches).

Directionality

In the original versions of Merlin the only directionality was a universal urge to understand. Built into the actions was the view that more specific knowledge was preferred, so that the system would operate to increase its understanding in whatever way it could. When it understood that it should make an external response, it would.

In Merlin 3 we propose a general approach to directionality. A goal is a β-structure. As is well known (Newell and Simon, 1972, Chapter 14) the use of explicit goal structures provides several functions: the definition of what is wanted; the memory that such a situation was wanted so control can return for retrying; a node for associating fruitful methods; and a place for the history of attempts to guide the next attempt. We will only be concerned here with the β-structure as a criterion for attainment of the goal.

The ingredients that are always present in a goal-ish situation are a goal G and a current situation, S, both described by β-structures. The first principle is:

> ***Directionality Principle:*** ***To achieve goal situation G from current situation S, attempt to view S as G: S/G.***

The result of applying this is either to produce no result at all (S/G cannot be solved) or to produce:

S: [G S1/G1 S2/G2 ... Sn Sn+1 ...] (56)

In this latter case, the present situation is the solution, providing the sub-mappings

Si/Gi ... are acceptable. These sub-maps may be acceptable in one of two ways: (1) Si/Gi simply succeeds, producing a new β-structure: [Gi ...] which is the mapped object or (2) Si/Gi is permitted to succeed by posit. Since mapping-by-posit is so universally applicable, each attempted map is under some type of posit restriction, e.g., one of the following:

> no posits permitted.
> at most one posit permitted, not at top level, over all sub-maps.
> at most one posit permitted, at top level of each sub-map.
> use only the explicit posit: X/Y.
> use only those posits required to effect W/Z.
> ...

The Directionality Principle in itself is sufficient for solving many problems. We can capture this in the following proposition:

> ***Natural Method Proposition:*** ***To understand the current situation relative to the goal is to know how to attain the goal.***

Of couse, such a situation only sometimes obtains. The domain in which the proposition is true we call the ***domain of natural intelligence.*** It is that class of situations where the ability of a system to assimilate (i.e., to create mappings) makes it apparent to the system what needs to be done to solve the problem. Related to this is the following conjecture:

> ***Weak-Method Conjecture:*** ***The so-called weak methods (Generate and Test, Heuristic Search, Hill Climbing, Match, Means-ends analysis ...) arise as Natural Methods.***

Let us explain this by means of a simple example. Figure 9 shows a typical means-ends analysis sequence taken from GPS (Ernst and Newell, 1969). The original goal is to transform one logic expression (PvQ) into another (¬P⊃Q). GPS has an interlocked set of methods that leads to the solution and embodies what is commonly called means-ends analysis. (PvQ) is matched against (¬P⊃Q) and a difference is derived (here DELTA(C): v → ⊃). This difference is used to set up a reduce goal, whose method is to select out of the set of available operators, one whose operation is defined as changing connectives (DELTA(C)). This selection is done via a mechanism called the Table of Connections which holds pre-established associations between differences and operators. Then the goal of applying the operator (R3 in this case) is created. It succeeds yielding a new expression. The process iterates, though in this case the solution has been found so that termination occurs right after the next match process. (The selection of R2 would have led to backtracking eventually.) In the scheme some aspects become goals (transform, apply, reduce) but not others (match, select). To each goal a distinct action scheme exists (the method), which embodies whatever rationality the means-ends analysis scheme possesses. These methods themselves are simply given.

```
Transform (PvQ)  into  (¬P⊃Q)
     Match (PvQ)  to  (¬P⊃Q) →   DELTA(C):v → ⊃
     Reduce DELTA(C)  on  (PvQ)
           Select rule R3 with function   DELTA(C) → R2, R3
           Apply R3 to (PvQ) → (¬P⊃Q)
     Transform (¬P⊃Q)  into  (¬P⊃Q)
           Match (¬P⊃Q)  to  (¬P⊃Q) → Satisfied

R1:  (Av(BvC)) → ((AvB)vC)
R2:  (AvB) → ¬(¬A ∧ ¬B)
R3:  (AvB) → (¬A⊃B)
```

Figure 9: Means-ends Analysis Sequence as it Occurs in GPS

We envision in natural intelligence that the attempt to see the current situation as the goal (i.e., the desired situation) leads to a mapping, i.e., the construction of a β-structure. This mapping may be inadequate but makes apparent the potentialities for manipulating the given situation so that it can be viewed acceptably as the desired situation. This happens only if the situation is sufficiently transparent, but it serves to solve many simple problems that are embryonic examples of a wide diversity of general methods. The initial situation must encode the total task structure, for the attempt to assimilate must encompass the means for transformation and the constraints, as well as the initial object.

In Figure 10 we show an example of how Merlin might solve this same problem. S expresses the total given situation: an initial expression plus the specification that one RULE can be applied of the three available. G represents the goal situation.[i] Each RULE is expressed in several alternatives. For our example, we will assume that there are at least three: one explicitly as a RULE and two as generic expressions, one corresponding to an element of the range, the other to an element of the domain. (We will return to this multiple representation below.) To keep matters simple, the initial situation S is limited to a single application of a rule. [ii] The initial step (and in a sense the only step) is to try to view this initial situation as a further specification of the desired situation.

Flat and Cover applied to S/G first leads to reducing the two expressions to the same schema (EXPR), which represents the connectives as components: [CONNECTIVE v] and [CONNECTIVE ⊃] (which simply amounts to selecting the appropriate principal alternative). The most obvious attempt to find a cover involves mapping each of the

[i] For clarity, we have omitted the structure which would be necessary to disambiguate assymetric arguments. In full form, G would look like: G: [⊃ [ANTECEDENT [¬ P]][CONSEQUENT Q]].

[ii] A complete specification would include that a composition of rules can be applied.

```
Initial state:
     S: [[v P Q] [ONE-OF R1 R2 R3]]

Final state:
     G: [⊃ [¬ P] Q]

Allowable rules and some of their alternative views
     R1: [RULE [FROM [v A [v B C]]] [TO [v [v A B] C]]]
         [v A [v B C] [ELEMENT FROM]]
         [v [v A B] C [ELEMENT TO]]
         ...
     R2: [RULE [FROM [v A B]][TO [¬ [∧ [¬ A][¬ B]]]]]
         [v A B [ELEMENT FROM]]
         [¬ [∧ [¬ A][¬ B]][ELEMENT TO]]
         ...
     R3: [RULE [FROM [v A B]] [TO [⊃ [¬ A] B]]]
         [v A B [ELEMENT FROM]]
         [⊃ [¬ A] B [ELEMENT TO]]
         ...

Additional knowledge about logic expressions
      v: [CONNECTIVE DISJUNCTION]
         [EXPR [CONNECTIVE DISJUNCTION]]

      ⊃: [CONNECTIVE IMPLICATION]
         [EXPR [CONNECTIVE IMPLICATION]]

View S as a G
    S/G → [[v P Q] [ONE-OF R1 R2 R3]]/[⊃ [¬ P] Q]

        → [EXPR [CONNECTIVE v] P Q [ONE-OF R1 R2 R3]] /
            [EXPR [CONNECTIVE ⊃] [¬ P] Q]

        → Cover #1: [CONNECTIVE v]/[CONNECTIVE ⊃] → v/⊃    Fail

        → Cover #2: [ONE-OF R1 R2 R3]/[CONNECTIVE ⊃]

            A(ONE-OF) → R1/[CONNECTIVE ⊃]  Fail
                         R2/[CONNECTIVE ⊃]  Fail
                         R3/[CONNECTIVE ⊃]  → [⊃ [¬ A] B]   Succeed

                      → S: [EXPR [CONNECTIVE v] P Q R3]

            A(R3)     → S: [⊃ [¬ P] Q]

    S/G → [⊃ [¬ P] Q] / [⊃ [¬ P] Q] → G   Succeed
```

Figure 10: β-structure Generation of Means-ends Sequence of Figure 9

components onto its correspondent, ignoring the [ONE-OF ...] component as unneeded. This fails to find an acceptable mapping, since v/⊃ is not possible (and no other component in S covers [CONNECTIVE ⊃], either). If the two expressions had in fact been identical, then this cover would have succeeded and the problem would have been solved (corresponding to a successful initial match in GPS).

The second try at a cover attempts to use [ONE-OF ...] to cover one of the components of G, say [CONNECTIVE ⊃]. We assume that ONE-OF has an action corresponding to the understanding of its semantics, namely that one of its components is to be an alternative. This produces the three mapping attempts shown, only one of which succeeds (R3). This leads to the formation of a new alternative for S, namely, one with R3 for [ONE-OF ...]. The existence of this alternative eventually generates the attempt to view it as a G. This leads to the execution of A(R3) (which we assume exists) which generates yet another alternative for S. When the mapping is tried with this alternative, it succeeds, solving the problem.

The above sequence depends on the existence of two actions, A(ONE-OF) and A(R3), which are necessary to define the task environment but do not in themselves provide a method of solving the problem. The mapping attempt involves, in addition to Flat and Cover, two principles: that alternatives should be tried, and that any actions involved (on schemas or components) should be executed. These principles are part of the general techniques of finding maps and do not have any special relationship to the means-ends method.

We have ignored numerous sub-maps which might be attempted and would subsequently fail (e.g., [CONNECTIVE v]/[¬ P]). It remains for an implementation of these ideas to demonstrate that the desirable sub-maps are sufficiently "obvious", and that the system will not be swamped by attempting an intolerably large collection of undesirable sub-maps.

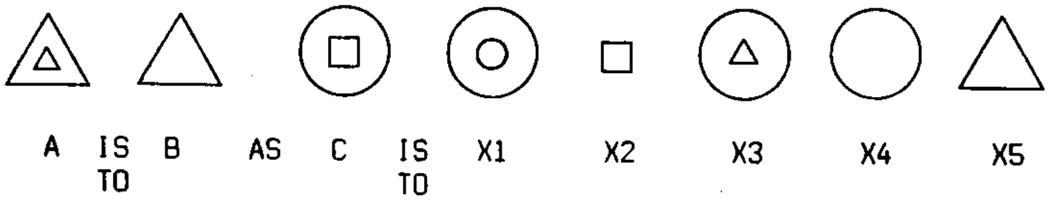
One feature of the task environment is closely related to the possibility of means-ends analysis. Each of the RULEs was described in terms of its output. That is, a RULE is a kind of an element of its range (recall the discussion of +). This is what permits easy selection of the RULE in terms of the goal. Without this (which corresponds to the prepared table of connections), there is no easy way to make the selection and the behavior of the system would correspond more closely to forward search.

This paper is not the place to attempt an extensive development of the notion of natural intelligence. However a second example will be useful to show its character. The top of Figure 11 gives a simple example of a geometric analogy problem, taken from Evans (1968) and by him from standard intelligence tests. Below the figure are written descriptions in β-structures. The task is indicated by the classic proportionality notation, which may be rendered "A is to B as C is to which X?". A direct translation into β-structures is:

[ONE-OF X1 X2 X3 X4 X5]/X (57)

where (58)
X: [C B/A*]

Original problem:



β-structure representation:

```
A:   [TRIANGLE LARGE [TRIANGLE SMALL INSIDE]]
B:   [TRIANGLE LARGE]
C:   [CIRCLE LARGE [SQUARE SMALL INSIDE]]
X1:  [CIRCLE LARGE [CIRCLE SMALL INSIDE]]
X2:  [SQUARE SMALL]
X3:  [CIRCLE LARGE [TRIANGLE SMALL INSIDE]]
X4:  [CIRCLE LARGE]
X5:  [TRIANGLE LARGE]

TRIANGLE:  [PLANE-FIGURE TRIANGLE-NESS]
CIRCLE:    [PLANE-FIGURE CIRCLE-NESS]
SQUARE:    [PLANE-FIGURE SQUARE-NESS]

X:  [C B/A#]
  → [C [A [[TRIANGLE SMALL INSIDE] NULL]]#]
  → [C [[TRIANGLE SMALL INSIDE] NULL]]
  (From the posit: TRIANGLE-NESS/SQUARE-NESS, we conclude
   that [TRIANGLE SMALL INSIDE]/[SQUARE SMALL INSIDE].)
  → [C [[SQUARE SMALL INSIDE]/NULL]]
  → [CIRCLE LARGE]

X/Xi → (fail) for i = 1, 2, 3 & 5.
But:
X/X4 → [CIRCLE LARGE] (success!)
```

Figure 11: Geometric Analogy Problem

That is, first form a view (X) of the fourth term, which is a kind of a C, namely the same kind of a C as B is a kind of a A (hence one wants the components of B/A, which define the mapping itself of B into A). Then, one wants that Xi which can be viewed as an X, so formed.[i] Figure 11 differs slightly, avoiding the use of ONE-OF and simply showing each of the Xi/X so that all mappings attempts are explicit. All of the intelligence in the selection is provided by the attempts to map with the basic Flat and Cover procedure.

The same situation prevails here as in the means-ends example. Simply expressing the problem in full in β-structures leads to a solution. Examination of the solution reveals two things. First, it depends on the encoding of the geometric figures, although the encoding used in no way smuggles in the solution. Second, actually finding the right answer depends strongly on the simplicity of the problem. Given a slightly more complex analogy or one in which the Xi were much closer together, the mapping could easily fail. However, this is exactly the point: Natural intelligence carries just a little way; to go further requires deliberate application of methods under the control of disciplined intention. This division of problem solving ability is essentially orthogonal to the one current in artificial intelligence programs, which produces expertise along certain classes of problems with complete inability to deal with any members of other classes, no matter how trivial.

Let us summarize. The act of attempting to view one entity as another is taken as the elementary act of directionality (of goal orientation). This implies not only a way of expressing goals when required, but that whenever Merlin performs X/Y it is operating in a directed way. The attempt to view one thing as another permits Merlin to solve simple, sufficiently obvious problems. The example of Figure 10 shows this for a simple example of means-ends analysis. The basic mapping technique in interaction with the structure of the task environment produced the equivalent of the methods of GPS. The point, of course, is not that there is no "method" -- as if the behavior arose magically. Rather, the single method of attempting to find a mapping suffices to generate different methods as a function of the structure of the task and goal. We have not demonstrated this yet for all the weak methods, but it appears to be a reasonable conjecture. There are limits to what the mapping mechanisms can do, in comparison to encoded methods which serve to maintain the directedness of solution attempts. Merlin must have methods for solving difficult problems, but we expect Merlin to rely on its natural intelligence (as embedded in the mapping) for a great diversity of trivial problems.

The above discussion is addressed to obtaining directionality. It does not address the Keep-progressing issue. In fact, natural intelligence will often fail and provide no inkling of what to do then. A clue can perhaps be found in the difficulty workers in artificial intelligence have always had in constructing spaces of methods. AI programs usually consist of a very few methods (less than the fingers of one hand); occasionally there are a dozen, each hand-crafted and pre-existing. Never is there a

[i] This is not the only way to view the problem; one could also find X: [B C/A#]. Or, one could represent each Xi as a C, separately (Xi/C) and compare these with the representation of B as an A (B/A) for similarity. Each is a bonafide kernel of a method and sometimes one, sometimes another is easier.

space from which one might generate method after method. Perhaps the attempt to find a structured space is misguided. Rather, the system should capitalize on the odd collection of methods that happen to be available within the system at a given time. Methods that have been used for other tasks must then be adapted to the task at hand, i.e., metaphor and analogy form the basis of the method space.

To be more specific, suppose Merlin has a method (M1) which can be confidently evoked whenever a particular problem (P1) is to be solved. That is:

[P1 M1]/SOLUTION (59)

succeeds. What happens when Merlin is posed with a different, but similar problem (P2)? Assuming that there is no existing method available which is known to be effective on P2, how might Merlin construct such a method? In the same spirit as the examle of Figure 11, above, Merlin constructs:

M2: [M1 P2/P1 *] (60)

That is, M2 is the image of M1 under that mapping which carries P1 into P2.

In Figure 12, we see a simplified example of this strategy in operation. TOWERS-OF-HANOI is intended to be the classical problem, here with five disks. The variant, TOWERS-OF-WARSAW has three disks and five pegs, but is otherwise the same. For M1, we have M-TH, a form of Generate-and-Test, parameterized by several aspects of the problem. In attempting to create a method, M-TW, for the TOWERS-OF-WARSAW, Merlin would proceed, as indicated above:

M-TW: [M-TH TOWERS-OF-WARSAW/TOWERS-OF-HANOI *] (61)

The similarity of the two problems provides a simple solution to the indicated submap, producing:

M-TW: [M-TH [TOWERS-OF-HANOI TW-PEGS/TH-PEGS
TW-DISKS/TH-DISKS] *] (62)

which is:

M-TW: [M-TH TW-PEGS/TH-PEGS TW-DISKS/TH-DISKS] (63)

Which amounts to saying "the method for solving TOWERS-OF-WARSAW is just the method for solving TOWERS-OF-HANOI with TW-PEGS replacing TH-PEGS and TW-DISKS replacing TH-DISKS.

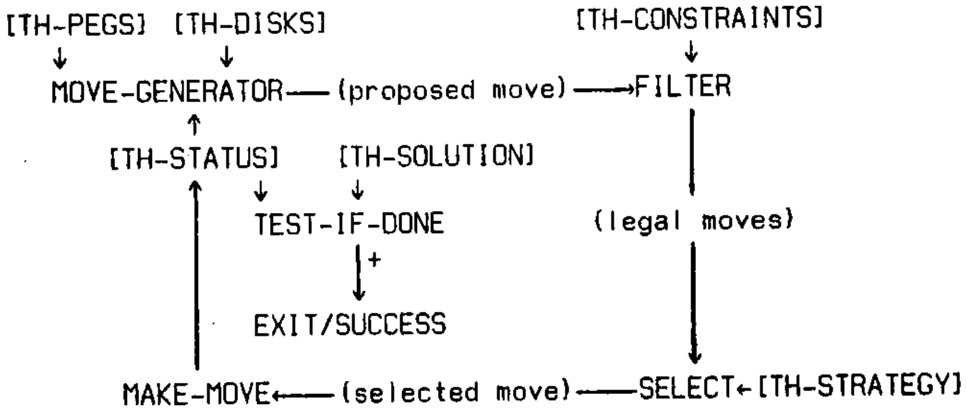
While this new method may not be optimal for its corresponding problem, it obviously has some chance of producing a solution. Also clear is that the complexity of the new problem or of the known method can be chosen to obfuscate this strategy.

Again, we have not shown that a genuinely large source of methods exists. We have generated an approach for Merlin to the issues of what to do next when you

A sketch of the definition of TOWERS OF HANOI.

```
TOWERS-OF-HANOI: [PROBLEM ...
                          TH-PEGS: [SET PEG1 PEG2 PEG3]
                              ...
                          TH-DISK: [SET DISK1 DISK2 DISK3 DISK4 DISK5]
                              ...
                 ]
```

M-TH, a method for solving TOWERS-OF-HANOI, represented as the outline of a flowchart:



A corresponding sketch of a variant problem: TOWERS-OF-WARSAW:

```
TOWERS-OF-WARSAW: [PROBLEM ...
                           TW-PEGS: [SET PEG1 PEG2 PEG3 PEG4 PEG5]
                               ...
                           TW-DISKS: [SET DISK1 DISK2 DISK3]
                               ...
                  ]
```

Figure 12: Towers of Hanoi and a Variant

don't know what to do -- find a similar problem and use it as a guide. Similarity is to be understood, not in terms of a metric on features of the problem, but in terms of whether one problem can be viewed as another so that the method for the first can be viewed as a method for the second. We might attempt to capture this design decision in a conjecture:

> ***Method-space Conjecture:*** *The set of methods used to continue on a hard problem (one where direct methods and natural intelligence have failed), is to be generated by viewing specific methods used successfully on other tasks as methods for the given task, more remote methods being generated as the mappings become weaker.*

Efficiency

The work on Merlin has in general not provided new design decisions on the question of efficiency. However the issue of cascaded interpretation is particularly critical for Merlin. Recall Figure 7 which showed a small part of the hierarchy of knowledge. If the only action that was available in Merlin was associated with PROCESS at the top of the hierarchy, then any attempt to assimilate, say, LTAT at the bottom must work through a cascade of assimilations -- to interpret LTAT it must be viewed as a further specification of LTA, but to interpret LTA it must be viewed as a further specification of LT, but to ... and so on. When PROCESS is reached the action must percolate back through all the intermediate structures to actually effect something at LTAT.

This problem is hardly unique to Merlin. Systems usually permit definitional hierarchies in which previously defined terms can be used in constructing new definitions. But in operation, most systems unwind the definitions right down to the primitives in order to interpret the definitions, and thus pose the problem of what has been gained for the system (as opposed to the specifier). With Merlin the problem is compounded since the unwinding is an elaborate act of interpretation.

Our approach to this issue is through compilation. We can encase this in a principal of continuous compilation:

> *A new action for a β-structure is to be compiled as a side effect of using that β-structure as part of an attempted map.*

As discussed earlier, every node of the Knowledge Net can have an action, which embodies the same knowledge as can be held by the β-structure. Thus, one should be able to create an action at a node given an adequate interpretation of the knowledge at that node. With respect to the hierarchy of Figure 7, assimilation of PS as a PROCESS leads to an action on PS; assimilation of HS as a PS leads to an action on HS; and so on. When LTAT is assimilated there is already an action on LTA, so that the cascade of interpretation is avoided.

The current versions of Merlin do a partial, but important, part of this continuous compilation strategy. Each action, A(X), can be considered to be composed of a pure procedure, P(X), plus a prototype memory, M(X), containing a set of variables and their associated initial values. When an action-less object (Y:[X Y1 Y2 ...]) is assimilated, A(X) is found, M(X) is copied into active memory and P(X) is executed. There is a point in time when P(X) has fully absorbed the componants of Y into the active memory and has performed all possible, relevant computations on them, prior to seeking information outside of Y itself. At this point, A(X)'s state of computation is fully represented in the current set of variables and their values in active memory. A copy of this set is assigned to be M(Y) and for P(Y), we use P(X), itself. Thus, when later Merlin encounters some use of Y (e.g., in Z: [Y Z1 Z2 ...]), A(Y) represents exactly the same information as would be derived by searching through Y to X, retrieving A(X), and executing it in the context of Y. The process as it stands never creates any new procedure code.

In the discussion of the design issues we raised a question about the role of formality in efficiency -- formality we claimed led to uniformity, hence (ultimately) to a lack of adaptation, which would translate into inefficiency. There are aspects of formalization which offset this difficulty, of course, such as the guarantees that specified processing leads to specified results, which could lead to more efficient processing.

The question of formalization has much deeper significance than just that of its effects on efficiency. In particular, it is a cliche of computer science that all systems that have been programmed are a fortiori formalized. This is supposed to follow from the nature of an instruction set as a discrete set of primtives with fixed rules for combination. Yet, while this is certainly a view, it need not be the only view, especially given that there are no models around for what an informal system would be like.

The principle of self-declaring components for Merlin provides a system that seems to have some of the properties of an informal system. There is no fixed set of rules whereby one can interprete a β-structure. The role it plays depends on an indefinite amount of context in which it is embedded, which depends on the alternative interpretations that are plausible given the totality of other β-structures, including its components. Thus, all interpretations are hypotheses. This would seem to be the essence of an informal system, though as we noted, there is no way to be sure. We can put this, then, as a conjecture:

Merlin is an example of an informal system.

Error

The design of Merlin offers nothing fundamental yet to cope with error. Merlin has no generalized back-tracking facility, such as exists in the Planner-like systems. The opportunities for combinatorial explosion appear so great in Merlin that backtracking simply cannot be afforded as the basework for dealing with error. Pending an alternative proposal, we have let the issue of error alone.

Depth of Understanding

The explorations with Merlin so far have not addressed themselves to questions of depth. In particular, Merlin does not now have a global data base, though it is planned to have one in the coming version.

Although not yet implemented in Merlin 3, the context related mechanisms (analogous to those used in earlier versions of the system) should be a powerful tool for addressing part of the access issue. In the course of attempting a map, Merlin may have cause to attempt a sub-map. Within this sub-map may be attempted still another sub-map, and so on, to an arbitrary depth. All of these pending maps are retained on a context stack, for use by any of the actions interested in the larger tasks which led to creating the current one. Furthermore, when the mapping mechanism is examining X for an action, if there is none then X↑, X↑↑, X↑↑↑, ... are examined, in turn. As each is stepped through to the next, the previous one is pushed onto the context stack. It is not worthwhile to explore here, all the intricacies of exactly how this stack is created, maintained and used. Suffice it to say that through this device, any action not only has access to all objects responsible for its evocation, but further, that they are all ordered on the basis of relevance--the top of the stack containing the most relevant objects, the bottom, the least. In prior versions, this machinery has enabled actions to reliably differentiate between similar objects in the computing environment which might otherwise be far harder to disambiguate.

CONCLUSION

Let us return to one of the reasons for organizing the discussion of Merlin around an extensive set of design features. An understanding system must come to terms with all the design issues in Figure 2. It is certainly possible to build systems that only respond to some of the issues. It has happened many times in the literature of understanding systems, and all of what we have done with Merlin to date is similarly limited. Our characterization of work on Merlin as explorations into design issues reflects that fact. Not until Merlin is prepared to respond to the several issues on which it is now silent -- multiple representation, accommodation, efficiency, error and depth -- will it be a full-fledged proposal for an understanding system.

The incompleteness of the design of Merlin. does not mean the existing design proposals are without content or merit, only that they are unstable. The attempt to solve the problem of error, which is absolutely critical, could modify the underlying attempt at informality with its implication of large amounts of uncertainty throughout the basic fabric of the system. Another example, though common to many systems beside Merlin, is the apparent inefficiency induced by the introduction of a general list-like data structure. These almost always run very slowly for large systems. Perhaps these difficulties can be overcome directly or through a suitable multiple representation capability. Another possibility is compiling of selected structures into more efficient data structures (arrays and bit-arrays). But if these are not effective, substantial change in the basic system could be called for.

Let us turn finally to the distinctive feature of Merlin -- taking assimilation as a central problem which dictates the representation of knowledge (the β-structure). The answer to the title of the paper -- How Can Merlin Understand? -- lies in this act of assimilation. Merlin will understand by the construction of maps from the structure that represents what Merlin knows to the structure that Merlin seeks to understand. The act of mapping is what will (hopefully) make Merln bring to bear the internal knowledge relevant to the task and in that respect to exhibit its understanding.

Several (conjectural) promises hang on this internal representation in terms of mapping. The most important, it seems to us, is the promise of a ***non-brittle*** system. Computer systems in general and artificial intelligence systems in particular have sharp boundaries between what they accept and what they don't. Even programs such as Winograd's (1971), which seem on the surface to be forgiving, remain highly brittle -- one passes quickly from what the program can handle (in terms of its English for instance) to where it fails completely. Only contentless systems like ELIZA (Weizenbaum, 1966) avoid this and they do so in ways that do not easily lead back toward intelligent systems. Attributing the difficulty to the basic discrete character of digital computation (which is surely right in part), one can seek to solve the problem by softening the basic representation. Merlin attempts a solution at a different point. Anything can be mapped into anything else, if sufficiently exotic maps are permitted, including those that posit some connections in order to infer others. Thus, the elastic character of the mapping holds Merlin's primary hope for avoiding brittleness. Since the representation is the mapping (an almost McLuhanesqe proposition) and since the representation is universal (i.e., for all types of knowledge, procedures, facts, etc.) the softness induced by the mapping applies to all aspects of Merlin's operation. We have seen, for example, how this might apply to methods to generate a much larger space of methods for a given task than is now available.

A second important point of the mappings is the possibility of Merlin being an informal system, in the sense described earlier. This is related to the question of brittleness above, but formality is an important issue in artificial intelligence in its own right. The considerations in the current paper are entirely preliminary and really only serve to introduce the issue.

There is no difficulty identifying the substantial problems that still exist in determining whether a general mapping scheme can be successfully embodied in a complete system. The generality of the mapping contributes substantially to the difficulties in obtaining good design proposals for the missing items on the list of design issues. The accommodation issue is an obvious example, as is the efficiency issue, but it is true of the others as well. There is also the problem, already mentioned, of how to control the mappings given that facilities exist for always succeeding. Thus we must put forward the design considerations of the present paper as still highly provisional.

REFERENCES

Anderson and Bower, *Human Associative Memory*, Winston and Sons, Washington, D.C., 1973.

C. G. Bell and A. Newell, *Computer Structures*, McGraw-Hill, New York, 1971.

B. S. Bloom, (ed.), *Taxonomy of Educational Objectives*, David McKay Co., New York, 1956.

D. G. Bobrow and B. Raphael, "New Programming Languages for AI Research", Tutorial Lecture given at the 3rd International Joint Conference on Artificial Intelligence, Stanford University, unpublished ms., 1973.

J. Carbonell, "Mixed-Initiative Man-Computer Instructional Dialogues", Ph.D. Thesis, Dept. of Electrical Engineering, M.I.T., 1970.

L. S. Coles, "Talking with a Robot in English", in D. E. Walker and L. M. Morton (eds.), *Proc. 1st International Joint Conference on Artificial Intelligence*, pp. 587-596, 1969.

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

T. G. Evans, "A Program for the Solution of Geometric-Analogy Intelligence Test Questions", In M. Minsky (ed.) *Semantic Information Processing*, pp. 271-353, M. I. T. Press, Cambridge, Mass., 1968.

P. A. Freeman and A. Newell, "A Model for Functional Reasoning in Design", *Proc. 2nd International Joint Conference on Artificial Intelligence*, pp. 621-640, London, 1971.

C. Hewitt, "PLANNER: A Language for Proving Theorems in Robots", in D. E. Walker and L. M. Norton (eds.), *Proc. 1st International Joint Conference on Artificial Intelligence*, pp. 295-301, 1969.

C. Hewitt, "Description and Theoretical Analysis (Using Schemata) of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot", Ph.D. Thesis, Dept. of Mathematics, M.I.T., 1972.

G. P. Huet, "A Mechanization of Type Theory", *Proc. 3rd International Joint Conference on Artificial Intelligence*, pp. 139-146, Stanford, Calif., 1973.

E. Hunt, "What Kind of a Computer Is Man?", *Cognitive Psychology*, 2, pp. 57-98, 1971.

R. L. London, "The Current State of Proving Programs Correct", *Proc. ACM 25th Annual Conference*, pp. 39-46, 1972.

J. Moore, "The Design and Evaluation of a Knowledge Net for Merlin", PhD. Thesis, Department of Computer Science, Carnegie-Mellon University, 1971.

J. Moses, "Symbolic Integration", Ph.D. Thesis, M.I.T., 1967.

A. Newell, "Heuristic Programming: Ill-structured Problems", in J. Aronofsky (ed.) *Progress in Operations Research*, *3*, pp. 360-414, John W. Wiley and Sons, 1969.

A. Newell, "Remarks on the Relationship between Artificial Intelligence and Cognitive Psychology", in R. B. Banerji and M. D. Mesarovic (eds.), *Theoretical Approaches to Non-numerical Problem Solving*, Springer-Verlag, Berlin, 1970.

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

N. J. Nilsson, "A Mobile Automaton: an Application of Artificial Intelligence Techniques", in D. E. Walker and L. M. Norton (eds.), *Proc. 1st International Joint Conference on Artificial Intelligence*, pp. 509-520, 1969.

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

J. Piaget and B. Inhelder, *The Psychology of the Child*, Basic Books, New York, 1969.

M. R. Quillian, "Semantic Memory", In M. L. Minsky (ed.), *Semantic Information* Processing, M.I.T. Press, Cambridge, Mass., 1968.

M. R. Quillian, "The Teachable Language Comprehender: a Simulation Program and Theory of Language", *Communications of the ACM*, *12*, pp. 459-476, 1969.

B. Raphael, "SIR: A Computer Program for Semantic Information Retrieval", In M. Minsky (ed.) *Semantic Information Processing*, M. I. T. Press, Cambridge, Mass., 1968.

B. Raphael, "The Frame Problem in Problem Solving Systems", in *Proc. Adv. Study Inst. on Artificial Intelligence and Heuristic Programming*, Menaggio, Italy, 1970.

J. F. Rulifson, J. A. Derkson, and R. J. Waldinger, "QA4: A Procedural Calculus for Inductive Reasoning", Stanford Research Institute, Technical Note 73, 1972.

R. Shank, "Conceptual Dependancy: a Theory of Natural Language Understanding", *Cognitive Psychology*, *3*, 1972.

G. J. Sussman and D. V. McDermott, "Why CONNIVING Is Better than PLANNING", M.I.T. Artificial Intelligence Laboratory, AI Memo No. 255A, 1972.

J. Weizenbaum, "ELIZA--A Computer Program for the Study of Natural Language Communication between Man and Machine", *Communications of the ACM, 9,* pp. 36-45, 1966.

T. Winograd, "Procedures as a Representation for Data in a Computer System for Understanding Natural Language", Ph.D. Thesis, Dept. of Mathematics, M.I.T., 1970.

FIGURES