THE UNIVERSITY of EDINBURGH

Analogy and Mathematical Reasoning
A Survey

C.D.F. Miller

M. Phil

University of Edinburgh

1982

CONTENTS

ABSTRACT

We survey the literature of Artificial Intelligence, and other related work, pertaining to the modelling of *mathematical reasoning* and its relationship with the use of *analogy*. In particular, we discuss the contribution of Lenat's program AM to models of mathematical discovery and concept-formation.

We consider the use of *similarity measures* to structure a knowledge space and their role in concept acquisition.

Preface

This dissertation is intended to constitute "a critical review of an area of the literature" of Artificial Intelligence[1]. It is perhaps necessary to justify why the at first sight diverse material treated constitutes an "area" worthy of treatment as a body. We shall hope to do this in the Introduction which follows this brief preface.

In AI at present it is extremely difficult to draw precise boundaries around clearly defined and distinct topics; a study of one part of the subject almost invariably draws the student into many other regions. Thus, what is presented here is not, and by the nature of AI can not be, a study of a self-contained field, but is rather an examination of a spectrum of AI literature with two main foci of attention: "mathematical reasoning" and "analogy".

We shall hope to convince the reader that in fact analogical reasoning and representation are central to cognition, and that in particular they are essential to mathematical deduction and discovery, which we consider paradigmatic examples of rational thought. Hence the intersection of these two topics, i.e. "Analogy in Mathematical Reasoning", can be seen as a microcosm of a

---

1 Henceforward usually abbreviated as "AI"

very wide range of cognitive activity; however, in order properly to understand the relationship between these two topics, it is necessary to consider each in a wider context.

In the Introduction we shall present arguments which will justify in more detail the choice of literature to be discussed in the succeeding sections. This will be followed by a review of a substantial body of AI literature. Finally, we shall outline in a very general way a possible model of the assimilation of information by analogy, making use of the notion of _similarity measures_.

Introduction

It is always hard to choose a title for a written text, and in view of this difficulty it is common to spend the first paragraph or so explaining what the work is *really* about. This survey is no exception to that rule.

Since the author's principal interest lies in the field of Artificial Intelligence, the greater part by far of the survey will be devoted to computational systems, implemented or proposed, which attempt to simulate various aspects of discovery and creativity in mathematics, and of the use of analogy in reasoning and knowledge representation. This does not mean that all the systems discussed take mathematics as their domain of activity; we shall, for example, consider a program to carry out IQ-test analogy recognition problems ([Evans1967a]), and programs to learn concepts by induction from examples [Winston1975a] and [Langley1978a]. The criterion for inclusion is that each system discussed should carry out (or attempt to carry out) some task of direct and immediate relevance to either or both of our principal topics, mathematical reasoning and analogy. As we shall hope to make clear in the rest of this introduction and in the subsequent sections, many such systems will be of much broader potential application than might be suggested by their apparent restriction to a relatively limited domain.

The choice of systems to be investigated will be seen to show two quite strong biases: for "analogy" systems, and against "conventional" theorem-provers. The reason for the pro-analogy bias is that we believe that the recognition and use of analogies is absolutely fundamental to any form of discovery, creativity or inductive reasoning. The reason for the exclusion of conventional theorem-proving programs is twofold: first, they are already extremely well studied in a number of sources (e.g [Nilsson1971a,Bledsoe1977a]); second, we believe that although deduction can play a significant role in discovery, the detailed differences among the internal mechanisms of particular deduction systems are of little relevance to our present work.

An essential component of reasoning is discovery; if we wish to produce a system which reasons intelligently, it is essential that it should be capable of learning from its previous experience. Thus much of our attention will be concentrated upon the notion of "mathematical discovery". This is a surprisingly hard concept to pin down accurately. It includes the proposal of new conjectures, if possible with some indication of the grounds for believing a conjecture, the strength of those grounds, and an idea of how to go about verifying or refuting the conjecture, the formation of new concepts, and the investigation of their properties; and the suggestion of interesting areas for potential further investigation by means beyond the present power or resources of the discoverer.

We shall begin our survey by considering some of the non-computational studies of discovery and creativity: Hadamard [Hadamard1945a], (the most directly concerned with mathematics), Koestler [Koestler1964a] and de Bono [Bono1967a]. We shall then, armed with a better understanding of what problems are to be addressed, consider the notion of Lakatos [Lakatos1976a] that discovery is implicit in the nature of proof and refutation, and examine one of the few detailed empirical accounts of mathematical creativity at work ([Waerden1971a]). Also to be considered at this point is the very important work of Polya ([Polya1945a], [Polya1954a], [Polya1962a], [Polya1965a]) on "heuristic".

After this foray beyond the computational world, we shall withdraw to ground on which the author feels his footing more secure, and study the literature of AI to see what has been achieved, what has been attempted, and what remains up to now neglected. The most important works we shall consider (i.e. important in relation to the present enquiry) are those of Lenat [Lenat1976a] on a model of discovery in mathematics, of R. Brown [Brown1977a] on the construction and use of analogies for transferring "expertise" from one domain to another, and of Munyer [Munyer1977a,Munyer1977b] on the use of analogy as a "fuzzy" matching rule, similar to unification, in a deductive system.

Before embarking on the critical part of the survey, we will state very briefly a few of our own views. To begin with, as already emphasised, a powerful analogy mechanism must underlie any discovery system. This is because a great deal of discovery stems from the drawing of generalisations from similar data, and in order for this to be possible there must first exist a criterion, and if possible a quantitative measure, of similarity between data. Furthermore, the very existence of any sort of taxonomic classification of the world requires the ability to discriminate between members and non-members of a class, whilst the fact that such classifications are, in the context of human learning, flexible and extensible necessitates a mechanism for acquiring new discriminatory criteria. It might be argued that the similarities used in such discrimination are simply in the form of the conjunction of common possession or non-possession of some set of characteristic properties, and that no more subtle analogy mechanism is needed; this argument fails on two grounds. First, it begs the question. How can it be determined whether an object possesses any given property? Even in elementary cases, where the property seems to be a simple perceptual datum, e.g. "yellow", the decision is not always clear-cut; yellow is only meaningful to most people as the common property of all things which they would call yellow[2] which seems to lead the above position into circularity. In more complex cases it is clear that we are in danger of being thrown into a regress of definitions that, even if not infinite, is so unwieldy as to become at the very least implausible as a discriminatory mechanism to be applied indiscriminately. Second, many concepts are not at all well-defined, and are extremely "context-sensitive" Thus it is very unclear whether any simple conjunctive definition is sufficient to define as diffuse a concept as "chair", or indeed any disjunctive definition of manageable size - "chair" describes a set of mutually similar objects, a set with rather fuzzy boundaries[3] (consider for example a doll's chair and a packing-case as dubious boundary instances). We shall explore this idea further in our discussions of

---

2 That the property does not have well-defined natural boundaries, which could for example be specified by giving a range of frequencies of light, is exemplified by the fact that the French "jaune", usually translated as "yellow", includes a range of colour which most English-speakers would usually term "brown" or "tan"; there is no French word corresponding precisely to the English concept "yellow". Hence the boundaries of a concept may be determined by convention (e.g. linguistic usage) rather than by an intrinsic common distinguishing property.

3 For a rather different approach to the whole question, the reader is referred to "The Republic" [PlatoBC360a], in which the problem of class membership is resolved in terms of partaking of the form of an appropriate frame, or "ideal" as it has been translated in the past.

[Lakatos1976a] and [Winston1975a]. The reader may compare the above discussion with Wittgenstein's notion of a "family resemblance" among a collection of objects [Wittgenstein1953a]; individual instances of a concept, according to Wittgenstein, overlap in a loose and unsystematic way, leaving the precise boundary of the concept unclear.

As will be clear from the foregoing remarks, we would hold that the need for a clear understanding of analogy goes far beyond the domain of mathematical discovery - in particular we see very close links with areas involving recognition of a datum as an instance of something familiar, such as the visual identification of objects. For example Shneier [Shneier1978a] has produced a visual recognition program whose mechanism he has shown to be in fact of considerable generality; as an example, he has used a version of the same program to correct spelling errors. It is clear in such a case that the processes involved in visual recognition are closely allied to those needed for feature recognition in general. We are limiting the main focus of our attention to mathematics merely in order to have for consideration an area of less intractable dimensions than the entire field of human cognition[4].

4 This is left as an exercise for the interested reader.

It should also be made clear that despite the emphasis which we have so far given to it, we do not believe that analogy alone is the key which will unlock all the secrets of intelligent behaviour. As we shall hope to point out during our analysis of other work, we would view an analogising and analogy-using system as forming only one component among many, albeit an essential one. Other serious problems which merit close attention include: the direction of attention towards significant tasks; the judgement of how best to perform those tasks (see especially our remarks on [Lenat1976a]); formal deduction (i.e. theorem-proving, although the connotations of the term are more closely bound up with purely mathematical activities than we would wish); knowledge retrieval ("memory"); and learning. Indeed, the list could be extended almost indefinitely. However, there seems to be one very important question which has up to now been addressed scarcely at all from a computational viewpoint, and which bears in some degree upon many of the facets of intelligence. how, given some new datum, can it be placed within the context of present knowledge? That is, how can it be determined what it resembles, and in what ways, and by how much[5]?

This, it should be noted, is a question both about the representations of present knowledge, and about the ways in which those representations are used in matching and retrieval.

This question may be borne in mind during the reading of this survey; although we shall find no complete answers, we may at least be able to pose some relevant further questions. It may be seen as a problem of assimilation - a new datum is to be taken and accommodated[6] within a framework of existing knowledge in some manner - or of retrieval ("reminding" as Carbonell [Carbonell1981a] calls it) - those items within existing knowledge relevant to the new datum are to be extracted. It is clear that these are two sides of the same coin, and that any postulated mechanism for one has very strong implications for the other.

The idea has been proposed of using similarity measures on graphs as a measure of the strength of an analogy, e.g. in [Potschke1982a]. There has not, I believe, been any discussion of the use of such measures in the search mechanism for analogy discovery. We shall therefore conclude this survey by making some tentative suggestions for an approach towards achieving this.

6 The term is taken from the theories of Piaget [Piaget1954a] on developmental psychology.

Non-computational Work on Analogical and Mathematical Reasoning

Before moving on to the main body of this thesis, we shall discuss briefly a few of the more important studies of discovery and creativity in mathematics and related fields which do not attempt to provide any sort of computational model of the phenomenon which they examine. We consider two categories of work: informal and anecdotal discussions, exemplified by Koestler [Koestler1964a] and de Bono [Bono1967a], and attempts at a more theoretical treatment (Polya [Polya1945a], [Polya1962a], [Polya1965a], [Polya1954a], and Lakatos [Lakatos1976a]).

In his book "The Use of Lateral Thinking" [Bono1967a], de Bono attempts to contrast "vertical" with "lateral" thinking; neither term is given a formal definition, but the former may be best described as analytical reasoning attempting to find a direct logical path from problem to solution, whereas the latter involves the deliberate seeking of unexpected solution paths (cf. the famous quotation of Souriau, "Pour inventer il faut penser à côté", cited on p.145 of [Koestler1964a]). De Bono's views are well summarised in the following extracts:

> "Vertical thinking has always been the only respectable type of thinking; in its ultimate form as logic it is the recommended ideal towards which all minds are urged to strive, no matter how far short they fall. Computers are perhaps the best example. The problem is defined by the programmer, who also

indicates the path along which the problem is to be explored. The computer then proceeds with its uncomparable logic and efficiency to work out the problem. The smooth progression of vertical thinking from one solid step to another solid step is quite different from lateral thinking". (P11)

"One of the techniques of lateral thinking is to make deliberate use of this rationalizing facility of the mind. Instead of proceeding step by step in the usual vertical manner, you take up a new and quite arbitrary position. You then work backwards and try to construct a logical path between this new position and the starting point. Should a path prove possible, it must eventually be treated with the full rigours of logic. If the path is sound, you are then in a useful position which may never have been reached by ordinary vertical thinking. Even if the arbitrary position does not prove tenable, you may still have generated useful new ideas in trying to justify it". (P12)

"New ideas depend on lateral thinking, for vertical thinking has inbuilt limitations which make it much less effective for this purpose". (P13)

It may be noticed that in the first of these quotations he asserts that lateral thinking is fundamentally non-algorithmic, whilst in the second he attempts to outline an algorithm for it! Indeed, it would be trivial to incorporate in any conventional problem-solving computer program some heuristic such as "choose an arbitrary fact and attempt to incorporate it into a solution path". However, the value of such a strategy is at best unclear; what would be required in addition is a mechanism for quickly reviewing a large number of possibilities and deciding which facts are candidates for further consideration. On this subject, consistent with his view that lateral thinking is essentially not amenable to an algorithmic definition, de Bono has nothing to say.

A number of writers have undertaken empirical studies of how creative thinking occurs in practice, in contrast to de Bono's prescriptive account of how it *ought* to occur; these are invariably founded upon introspection by creative thinkers, or, to be more accurate, on subjective *post hoc* reconstruction of introspection into the creative process.

A pioneering work of this kind was the monograph by Hadamard, "The Psychology of Invention in the Mathematical Field" [Hadamard1945a]. In this he discusses the examples of Poincaré (quoted from "Mathematical Creation" [Poincare1913a]), Kekulé's discovery of the benzene ring, and a large number of other examples of "inspiration" among well-known mathematicians. However, his proposed "mechanism" for such creativity draws heavily on the unelaborated workings of the "unconscious mind", where he supposes that very many ideas are combined essentially at random, becoming accessible to consciousness (i.e. introspection) only when a fruitful combination is found. What constitutes an "idea" or a "combination" is left undiscussed, and there is no consideration of how many attempted combinations may be required, nor how much processing is to be done on each combination. It is apparent that it is necessary to postulate either a vast capacity for unconscious processing or else a filtering mechanism to enable only plausible combinations to be examined, if one makes the reasonable assumption that the number of "ideas" in the memory available for combination is large. Whilst it is perhaps possible to extract the germ of an algorithm from Hadamard's imprecise theory, it is clear that without an efficient mechanism for search control and selectivity, no practical computer program could be produced to run on the hardware of the foreseeable future which would embody this theory. Indeed, it is clear that Hadamard was concerned rather to present a phenomenological account of the process of mathematical discovery than to provide a theory of its mechanism which would be testable and refutable.

Another major study in this area is Koestler's "The Act of Creation" [Koestler1964a], in which the horizons are broadened from mathematical discovery to creative thinking in general. Once again, much is left to the mysterious, apparently non-algorithmic workings of "the unconscious", reinforced by the examples of Poincaré, Kekulé, Ampère, Gauss and Hadamard (pp.116-118). However, Koestler does lay great stress on the essential role of analogy in the creative process; he coins a new term, _bisociation_ to describe "perceiving of a situation or idea, L, in two self-consistent but habitually incompatible frames of reference" (p.35), and goes on to develop this into the idea of constructing an analogy via L between these two frames of reference. He later makes the unequivocal assertion that "discovery consists in seeing an analogy which nobody had seen before" (p.104), and later (p.120) that "[the creative act] does not create something out of nothing; it uncovers, selects, re-shuffles, combines, synthesizes already existing facts, ideas, faculties, skills. The more familiar the parts, the more striking the new whole". However, he regards this process of selection, re-shuffling, etc. as being essentially non-algorithmic:

> "Here, then, is the apparent paradox. A branch of knowledge which operates predominantly with abstract symbols, whose entire rationale and credo are objectivity, verifiability, logicality, turns out to be dependent on mental processes which are subjective, irrational, and verifiable only after the event." (P.147)

> "The search for the improbable partner involves long and arduous striving - but the ultimate matchmaker is the unconscious." (P.201)

It is, of course, an article of faith amongst AI researchers that the mystic unconscious processes invoked by Koestler and Hadamard are modellable as computational processes, such faith must ultimately be justified by exhibiting appropriate models, and it is the attempts to do so, or to take steps towards doing so, which will form the subject matter for the remaining sections of this thesis.

Before leaving this review of non-computational studies we should, however, consider some attempts to formalise rather more explicitly aspects of mathematical discovery.

The most substantial and best-known of these is the work of Polya [Polya1945a, Polya1954a, Polya1962a, Polya1965a]; in the earliest of these texts, "How To Solve It", he presents what is essentially a dialectic approach to problem-solving in which the problem-solver asks himself a series of questions to guide his search, and to reveal possible alternative approaches, e.g. (pp.xvi-xvii)

> "Do you know a related problem?
>
> "Here is a problem related to yours and solved before. Could you use it?
>
> "Can you use the result, or the method, for some other problem?"

In the later, more substantial, works Polya presents a large number of detailed examples, from which he abstracts further general maxims. A principle akin to Koestler's idea of bisociation is abstracted from examples in geometric construction and subsequently widened very generally, viz. finding two "loci" for the solution to a problem and then finding their "intersection" [Polya1962a, ch.6][7]. However, the overall treatment which emerges from his work is still primarily anecdotal and unsystematic. The heuristics discussed are described informally, and frequently, as in the above instance, analogically.

Whereas Polya's examples are generally "rational reconstructions" of how a discovery could be made, an interesting example is given by Van Der Waerden [Waerden1971a] of how a proof was actually discovered in practice by a group of mathematicians in discussion. Plotkin [Plotkin1977a] has suggested that some of the discovery steps illustrated in this paper might be amenable to inclusion in a very advanced theorem-proving system; certainly some of the heuristics - "try to obtain a stronger form of the theorem", "try to generalise the theorem", "use the strongest induction hypothesis possible" - are extremely valuable in mathematical proof. Some of them have indeed been incorporated into systems such as those of Cohen [Cohen1980a] and of Boyer and Moore [Boyer1979a]; however, the sophisticated application of these described by Van der Waerden still seems to be beyond the scope of present programs.

One possible route which work such as Polya's and Van Der Waerden's might indicate is the development of rule-

---

7 It is interesting to notice in passing how analogy here puts in an appearance in the description of a problem-solution method.

based expert systems to incorporate the heuristics used by practising mathematicians. Lenat's program AM [Lenat1976a] can be seen as a step in this direction; however, as we shall argue below, AM's "heuristics" are too low-level and the deductive power of the system too weak to be regarded as a true expert system, although Lenat's contribution is valuable in other respects. Perhaps more fruitful than a self-contained program such as AM would be an interactive system incorporating a proof engine and an expert adviser, the latter proposing directions of exploration to a user who could then use the former to test the consequences of those suggestions which seemed most potentially fruitful. One eventual goal of such a system would of course be the extraction and formalisation of the user's expertise for incorporation within the expert system itself, in the tradition of "knowledge refinement" as propounded by Michie and others.

It is interesting to compare this suggestion with that of Michener [Michener1978a]. She attempts to define a detailed structure of mathematical knowledge (having many similarities with the structures used in the CAI work of Pask et al [Pask1975a]), dividing it into _examples_, _results_ and _concepts_, with many further subdivisions and cross-links between these. She then proposes an interactive computer system which will "help neophytes understand mathematics and learn _how_ to understand" by guiding them through the knowledge-base. Finally, she suggests using this system in conjunction with "theorem provers, or analogy- or concept-generating programs that need to use previously established mathematics". The examples given of the sort of advice the system might give to a non-resolution theorem-prover look very similar to some of Polya's heuristics.

We conclude this brief survey of non-computational studies of mathematical reasoning by considering the work of Lakatos [Lakatos1976a], whose approach is substantially more formal and more philosophical than any of the works discussed above. Lakatos is strongly influenced by the ideas of Popper [Popper1959a] on the nature of a scientific theory, and of empirical induction. Briefly, Popper's view is that a theory is only meaningful if it is <u>falsifiable</u>, i.e. in principle refutable as a consequence of some experiment or observation. For if a theory is <u>not</u> falsifiable, then it tells us nothing of substance about the world; like the unobservability of the lumeniferous ether, it makes no difference to our predicted observations of events in the real world whether or not the theory holds. In "Proofs and Refutations" (the title an obvious parallel with Popper's "Conjectures and Refutations" [Popper1963a]) Lakatos extends the idea of empirical theory formation to a domain not normally regarded as empirical, namely mathematical proof. He illustrates how a theorem (e.g. Euler's relationship between faces, edges and vertices of a polyhedron) implicitly defines a collection of objects for which the theorem holds, and how failed attempts to prove the theorem may lead to a more precise definition of the appropriate concept. Thus a particular proof of Euler's relationship may fail for a certain class of polyhedra with "holes" in; hence the new concept of a "simply-connected" polyhedron is introduced. This process of alternately refining a concept definition and re-working a proof has much in common with Young, Plotkin and Linz's "rational reconstruction" of Winston's work on concept formation [Young1977a,Winston1975a], to be considered in greater detail below, in which a concept is considered to have a "least upper bound" and a "greatest lower bound", i.e. a pair of definitions one of which is sufficient and the other necessary. The process of concept-formation consists of pushing these bounds closer together until (perhaps) they coincide[8]. Thus in Lakatos' example, at any stage of his dialectic process we can determine of most objects either that they definitely are, or that they definitely are not polyhedra; however, there is a certain class of objects about which our current definitions leave us uncertain. In addition to the above mentioned work of Young, Plotkin and Linz, this model of

8 Of course, they need never merge - this may well be one way of capturing the "fuzziness" inherent in many concepts which was pointed out in our introduction.

concept formation and representation has been studied by Mitchell [Mitchell1978a].

As presented by Lakatos, the notions of proof and of concept formation are seen to be dual aspects of mathematical discovery. It is therefore appropriate that we begin our survey of AI work on mathematical reasoning with a survey of work on concept formation. As we shall see, this has close links with the formation of analogies.

Models of Concept Formation in AI

This is an extremely broad area, and we can only touch here on a few of the most important or most relevant examples. One major piece of research in this area, Lenat's program AM [Lenat1976a], is deferred to a later section for more detailed consideration, since it is especially important to our study of mathematical reasoning; the papers considered in the present section belong to the wider area of concept-formation in general, rather than being limited to the specific domain of mathematics.

Following our remarks at the end of the previous section on Lakatos [Lakatos1976a], we begin with an examination of the very well-known work of Winston on "Learning Structural Descriptions from Examples".

Winston's Structural Description Learning Program

In this discussion of Winston's concept-learning program [Winston1975a] we shall not be concerned with the initial scene-analysis component of the program, in which a descriptive network is obtained from a "blocks world" scene. Rather, we shall be concerned with the way in which successive "examples" and "near-misses" are used to refine the definition of the concept to be acquired. In an attempt to clarify terminology, we shall use the term "example" to mean "any scene presented to the program" (this is <u>not</u> Winston's usage), "instance" for "an example which satisfies the concept's definition", and "non-instance" for "an example which fails to satisfy the concept's definition".

Winston's representation of scenes as networks of nodes and arcs appears quite simple and natural for elementary scenes (e.g. his figures 5.5, 5.6 on p.161). However, as scenes become complex the networks become correspondingly unwieldy, and the set of "primitives" used to label nodes and arcs becomes both large and seemingly arbitrary.

His use of the same network structure with some additional primitives to represent concepts is, at best, confusing. Within a single network there are typically segments representing particular objects, segments describing relations between them, and segments describing properties of these relations (such as the "MUST-BE-SUPPORTED-BY" arc of Winston's figure 5.8). This flattening of a conceptual hierarchy into a single uniform structure does not aid clarity; nor does it appear to enhance the power of Winston's formalism - if his claim to be able to handle further recursive levels of abstraction is indeed justified, then his representation would surely be enhanced by a more obviously hierarchical structure (although such structure can of course always be superimposed by the reader upon the "flat" networks, given sufficient effort).

For a detailed criticism of Winston's work, and particularly of his representations and their amenability to the algorithms he describes, see the review by Knapman [Knapman1978a], which casts some doubt upon whether the mechanisms and representations described by Winston are in fact fully sufficient for the tasks which he claims that his program could carry out. A sympathetic account of Winston's thesis is given as clearly and succinctly as seems feasible by Boden [Boden1977a].

For the rest of this discussion, we shall concern ourselves with what seems to be a fair abstraction of the essence of Winston's learning mechanism, already mentioned above in our discussion of Lakatos. Young, Plotkin and Linz [Young1977a] have produced a "rational reconstruction", implemented as a POP2 program, of the use of instances and near-misses to learn a conjunctive concept; more recently Bundy [Bundy1981a] has produced a short Prolog program which embodies this model. The (strong) presupposition which underlies this model is that the set of possible attributes of objects and their relationships is arranged as a collection of well-defined, already known hierarchies.[9] The question of how the concepts embodied in these hierarchies are themselves acquired is not

---

9 Lattices in the Young-Plotkin-Linz model; we shall describe the model using hierarchies with an added "bottom" element, and then go on to observe how it may be extended trivially to general lattices.

considered by the authors; as we shall note below, their acquisition would appear to require a mechanism for the learning of disjunctive concepts.

A simple example of a hierarchy might be a SHAPE hierarchy:

```
            SHAPE
            /   \
      PYRAMID   PRISM
                /   \
           WEDGE    BLOCK
                    /   \
                CUBE    CUBOID
```

where entries in the tree are subsumed by their "parent" nodes.

In Bundy's program a concept is represented as a collection of constants which represent the objects composing the "ideal" instance of the concept, together with a set of property hierarchies, each with a pair of nodes marked as "upper bound" and "lower bound". Each hierarchy is associated with some object or group of objects in the "ideal", and represents the assertion that all the corresponding properties of the objects in any instance must lie below the upper bound, and that in any non-instance at least one attribute will lie above the appropriate lower bound. That is, the upper bounds give <u>necessary</u> conditions for the concept, while the lower bounds give <u>sufficient</u> conditions. A very similar model is embodied in Mitchell's concept of <u>version spaces</u>

[Mitchell1978a], in which sets of rules are retained giving the most specialised definition (lower bound) and most general definiton (upper bound) so far found to be applicable to a concept.

When an example is presented, the first task is to match the objects of the example with the constants of the ideal. This is done on a somewhat ad hoc basis by both Winston's program and by Bundy's rational reconstruction; as Bundy has pointed out (private communication) there is obviously much scope for a clever matching algorithm to find the "best fit" between example and ideal. Such an algorithm would in effect be an analogy-finder of the sort which we shall see is required by any program which discovers and uses analogy. When a match has been made, by whatever means, the property hierarchies of the concept definition are compared with the attributes of the example. For each property there are three possibilities:

1) The example lies below the lower bound; in this case one component of the conjunctive definition of the concept is satisfied. If <u>all</u> the relevant attributes of the example fall below the corresponding lower bound, then the example is an instance of the concept;

2) The example lies above the upper bound; thus it fails to satisfy one component of the definition, and must

be a non-instance;

3) The example lies in the "grey" area between the two bounds; in this case the program has the opportunity to refine its definition by adjusting either the upper or the lower bound, according as the example is a non-instance or an instance (the user provides this information).

A point not observed by Bundy in the cited paper about case 3) is that a non-instance is useful only when precisely <u>one</u> attribute falls in the grey area; otherwise the program does not know which upper bound should be lowered. It is for this reason that the choice of training sequence is critical; we shall give an example below where the same example may provide different information at different stages in the teaching sequence. It may also be noted that whereas a non-instance can lead to the adjustment of at most one upper bound, an instance may potentially lead to the simultaneous adjustment of <u>all</u> lower bounds, i.e. instances seem to convey more information than non-instances, in general. This point is considered in the survey paper by Bundy and Silver [Bundy1982a], where three cases are distinguished: instance, near-miss and far-miss (the last being the case when more than one attribute is "grey"). As alternatives to the conservative strategy of drawing no information from a far-miss, they offer the possibilities of either choosing an arbitrary attribute and lowering its upper bound, or lowering the upper bounds of <u>all</u> grey attributes. It may easily be seen that each of these strategies may lead to incorrect definitions being formed, so that the system must record any such decisions and be prepared to backtrack over them if it finds it has made a wrong choice.

As an example, consider the case where the concept to be learned is BLUE BLOCK, given a single object and the following two property hierarchies:

```
              SHAPE
              /   \
       PYRAMID     PRISM
                   /   \
              BLOCK     WEDGE
              /   \
          CUBE     CUBOID


               COLOUR
              /      \
        PLAIN          PATTERNED
        /   \          /       \
    BLUE    GREEN   STRIPED    DOTTED
    /  \
NAVY    AZURE
```

Initially we have upper bounds (SHAPE,COLOUR) and lower bounds (BOTTOM,BOTTOM) - we introduce an arbitrary element BOTTOM into each hierarchy to convert it into a lattice with lower bounds always well-defined.

The following table shows a learning sequence of examples, with the consequent revision of upper and lower bounds. The revised bounds at each stage of the sequence are shown underlined.

Example	Instance?	New bounds Upper	New bounds Lower
-	-	(SHAPE,COLOUR)	(BOTTOM,BOTTOM)
AZURE CUBE	Yes	(SHAPE,COLOUR)	(CUBE,AZURE)
STRIPED CUBE	No	(SHAPE,PLAIN)	(CUBE,AZURE)
NAVY WEDGE	No	(SHAPE,PLAIN)	(CUBE,AZURE) *
AZURE CUBOID	Yes	(SHAPE,PLAIN)	(BLOCK,AZURE)
NAVY CUBE	Yes	(SHAPE,PLAIN)	(BLOCK,BLUE)
AZURE PYRAMID	No	(PRISM,PLAIN)	(BLOCK,BLUE)
NAVY WEDGE	No	(BLOCK,PLAIN)	(BLOCK,BLUE) **
GREEN CUBOID	No	(BLOCK,BLUE)	(BLOCK,BLUE)

At this point all upper bounds coincide with all lower bounds, so the concept has been learned. It is interesting to compare this procedure with that of Langley's program BACON.1 [Langley1978a], discussed below.

We can make several observations about the above model. First of all, it requires a prior knowledge of the property hierarchies; this is a serious weakness since a fundamental part of concept formation is precisely the acquisition of a conceptual framework within which new concepts are to be assimilated. However, the condition that the knowledge be organised as hierarchies can be slightly weakened; the technique of converging upper and lower bounds can clearly be used on an arbitrary lattice[10], should such a representation prove useful. It is

* The NAVY WEDGE gives no useful information at this point ...

** ... but here it does!

10 Note that we use "lattice" in the formal mathematical sense here, viz. a partially ordered set such that every pair of elements has a unique least upper bound

interesting to note at this point that, as we shall see later, a very similar criticism about a priori knowledge can be made of Lenat's concept-discovering program AM [Lenat1976a], which does indeed use a lattice to represent the organisation of its concepts. In Winston's original program, it is perhaps not completely obvious that the property hierarchies used by Young, etc. are explicitly present; they are in fact provided by his "A-KIND-OF" links, and our criticism can thus be applied to his apparently arbitrary choice of "primitive" nodes and their "A-KIND-OF" subsumption relations.

A second major criticism is that only conjunctive concepts can be learned by this method. It can certainly be argued that such concepts are in practice more common than disjunctive ones, and that people find disjunctive concepts relatively hard to learn. However, the counter-argument that the nodes of the property hierarchies are themselves disjunctive concepts (e.g. BLUE is AZURE or NAVY), and that these concepts must themselves be acquired, is hard to answer. This aspect of concept formation, which can be seen as the creating of generalisations, is clearly related to the processes which underlie analogy formation, since an analogy can be considered as a unifying generalisation of two disjoint concepts.

---

and a unique greatest lower bound.

We may also observe, with Knapman [Knapman1978a], that there is some doubt as to the psychological validity of a concept formation mechanism which relies heavily on the presentation of known non-instances. This point is related to the previous criticism of having a predefined space of attributes; without such an _a priori_ organisation of knowledge the idea of an "upper bound" of a concept would be meaningless.

Finally in our discussion of Winston, we note that the model does not make the fullest use of the information presented to it; when presented with an example (such as the first occurrence of NAVY WEDGE in our above scenario) where more than one attribute falls into the "grey" area, no progress is made at all, even though it would in principle be possible to record that one or other of the previously known upper bounds needed to be revised, and then to use subsequent examples to determine which - this would require that some "disjunctive" information be retained, contrary to the spirit of this model.

<u>Langley's program BACON.1</u>

In his paper [Langley1978a] Langley describes a "general discovery system", BACON.1, which gathers data and attempts to induce laws governing regularities therein. Thus he is dealing with a particular, simplified instance of the general problem of assimilating new knowledge to an existing knowledge-base - "simplified" because in his program, as in Winston's [Winston1975a], only a single "concept" is being assimilated at one time, and the program implicitly assumes that all input is relevant to this.

Such an assumption is justified in the case of BACON.1 because the program is not merely a passive recipient of "instances" and "non-instances", but instead acts as a data-gathering agent by asking of its environment (i.e. the user) what values the dependent values of a relation will take given a particular set of independent values chosen by the program. Thus BACON.1 performs experiments upon its environment in order to infer laws governing its structure.

We shall describe here two tasks performed by BACON.1, and then discuss how these are carried out.

The first task is the discovery of a simple numerical relationship: given the orbital distance $\underline{d}$ and period $\underline{p}$ of three planets, the program notices after examining successively $(\underline{d}/\underline{p})$, $(\underline{d} * (\underline{d}/\underline{p}))$, and $((\underline{d}/\underline{p}) * (\underline{d} * (\underline{d}/\underline{p})))$ that the last of these is constant. This is Kepler's third law of planetary motion. Since BACON.1 lacks any algebraic simplification rules, the final term above is not translated to $\left[\frac{d^3}{p^2}\right]$.

The second task is a simple concept-formation in the style of [Winston1975a]. The program is given three independent variables (shape, size and colour) and their domains of possible values, and one independent variable ("feedback"). It then asks for various values of feedback and receives the successive responses:

```
large  blue  square: no
small  blue  square: no
large  red   square: yes
small  red   square: yes
```

by which stage it has formulated the hypothesis

```
colour=red  => feedback=yes
colour=blue => feedback=no
```

which it then confirms by trying

```
large  blue  circle
small  blue  circle
large  red   circle
small  red   circle
```

The disparity between the above two tasks suggests that either BACON.1 does indeed embody some general principles of discovery or else it possesses a mixture of methods apt for various different tasks. We shall argue that, while both of these contain a measure of truth, the latter is in fact a more significant factor in the program's apparent versatility.

BACON.1 is a production-rule system whose rules fall into five categories, totalling in all about 75 rules:

Data-gathering - these govern the program's acquisition of "raw" data by means of a factorial experiment[10] in the independent variables. (Hence, the existing program is only suitable for handling variables with a finite domain).

Identity-checking - these check that algebraic combinations (called by Langley "higher-level attributes") of the independent variables, proposed as relevant by the regularity-checking rules, have not previously been examined in another guise. They thus prevent the program from looping, and are for "housekeeping" purposes only. In principle they could be replaced, to the benefit of the program, by a set of general algebraic simplification rules.

Attribute-evaluation - these obtain or compute the values of dependent variables, some of which are obtained from the environment while others are higher-level

10 I.e. an experiment in which all possible combinations of the independent variables are systematically examined.

attributes proposed by the regularity-checkers.

Regularity-checking - these look for regularities in the input data (e.g. two numeric attributes increasing together) and propose new attributes as candidates for testing (e.g. the ratio of two numeric attributes). As a special case, constancy of an attribute is recognised, and the discovery of a constant attribute may be regarded as the goal of the program.

Generalisation-testing - these check further data to determine whether a proposed law actually holds.

It is the fourth category, the regularity-checking rules, which are of principal interest to the present discussion since these embody the claimed general discovery mechanism of BACON.1.

Let us look at the planetary motion example more closely. Only two rules are needed to find the regularity here:

If two attributes increase together, consider their ratio.

If one attribute increases as another decreases, consider their product.

Clearly these two rules suffice for discovering any rule of the form:

$$\frac{A^a B^b C^c \ldots L^l M^m}{N^n P^p Q^q \ldots Y^y Z^z} = \text{constant}$$

A further rule proposes linear combinations of attributes, thus allowing BACON.1 to discover constancy in any rational function of the independent variables. There is another rule which looks for periodicity (so that BACON.1 can, for example, "explain" series such as 11121314151...) and proposes attributes of the form (a MOD b).

Rules of the above type are described as "trend detectors", and operate only on numeric data. The remaining regularity checkers are "constancy detectors" which work on either numeric or symbolic data.

It would appear that this singling out of numeric data seriously weakens Langley's claim of generality. However, matters are not quite as bad as might at first be assumed; in particular periodic regularities in symbolic data[11] are recognised, since the periodicity is itself derived from a numeric attribute (viz. position in the sequence). However, it is very hard to imagine how complex non-numeric concepts such as Winston's "arch" could be acquired by this sort of rule; one problem is that in harder tasks like this it is not feasible to provide the program with a small fixed set of independent attributes, each with a finite domain.

---

11 E.g. blue square, red circle, red square, blue circle, red square, red circle

This comment leads us to what is probably the most serious weakness of BACON.1. By restricting its domain to "toy" worlds where the number of possible different inputs is finite, indeed small, and free of "noise", Langley has avoided all the problems of search control. It is clear that if the program were, for example, to be able to recognise more elaborate numerical relationships (exponential, logarithmic, sinusoidal, square-root, derivative, etc.) the number of candidates generated by the regularity-checkers would rise very greatly. Similarly, if even as few as six symbolic attributes with six legal values each were defined, BACON.1's factorial experiment would require about 50000 sets of independent data to be supplied, and would obviously become intolerably large.

One notion which is lacking is any idea of hypothesis testing by the generation of crucial tests to decide between rival hypotheses. Armed with such a mechanism, BACON.1 could avoid performing the entire factorial experiment, and instead examine its hypotheses to choose those cases which might refute them. The only hypothesis testing described by Langley is, roughly, "if the hypothesis is seen to be true for the first four sets of data encountered then it is accepted"; as a principle of induction this would scarcely satisfy the most pragmatic of positivists, let alone disciples of Popper, and can scarcely be regarded as an accurate model of Baconian

Scientific Method!

In summary, we find Langley's claim for the generality of his program unconvincing. Furthermore, although his method produces quite elegant results in small domains, we doubt very much whether it could be extended to cope with large rule-sets, leading to very large search spaces where the number of alternative hypotheses would become unmanageable; it is also not easy to see how it could be made to handle noisy data, or data from continuous domains. Langley has failed to confront one of the central problems of discovery, and of AI as a whole, namely that of controlling search to defer the onset of the combinatorial explosion.

In the above discussions of the programs of Winston and Langley we have seen two widely contrasted approaches to concept acquisition, each with a number of shortcomings.

Perhaps the most immediately apparent distinction which may be drawn between the systems is the passivity of Winston's program contrasted with the positive data-gathering of Langley's; however, the latter is largely illusory since BACON.1 is in fact simply trying out every possibility within its search space; later versions, BACON.3 and BACON.4, behave rather more intelligently in this respect ([Langley1979a], [Bradshaw1980a]), as well as being able to cope with a small amount of "noise" in the input data, but do not differ very significantly from the model described above. A more genuinely active explorer of a non-trivial search space is Lenat's program AM, which we shall describe in some detail below. This program not only chooses which examples it wishes to study, but also generates the appropriate data itself. In this respect, AM shows the true beginnings of a concept-learning program. Furthermore, AM assimilates its concepts within the same structure as its prior framework of knowledge, whereas Winston's description "primitives" seem to belong to an entirely different category of knowledge from the concepts which they are used to describe.

Nevertheless, all of these programs are open to the question of where their initial knowledge-base derives from. In contrast, several workers have been working on the recognising of structure and pattern within input data with no prior collection of concepts or description primitives, e.g. Hedrick [Hedrick1976a] and Vere [Vere1977a].

<u>Other Work on Concept Formation</u>

There is a very large body of work on this topic, as remarked earlier; having considered what we regard as two paradigmatic examples, we shall not discuss the rest of this field in great detail - a thorough bibliography can be found in the SIGART special issue on machine learning

[Mitchell1981a].

This reports, amongst many other items of research, the work of Shapiro [Shapiro1982a] at Yale University, who claims to have developed a program which is capable of inducing, for example, the Peano axioms of arithmetic from facts such as

"0 < succ(0)" is true

"plus(succ(0),succ(0),succ(succ(0)))" is true

"times(succ(0),0,succ(0))" is false

Shapiro reports in his summary in SIGART that his model is based on Popper's methodology of conjectures and refutations [Popper1963a]. Bundy and Silver [Bundy1982a] give a brief summary of his technique of "contradiction backtracking", which discovers faulty rules. Unfortunately we have so far been unable to obtain further details of this interesting work.

In Mitchell's paper on generalisation ([Mitchell1979a]) the distinction is drawn between <u>model-driven</u> and <u>data-driven</u> strategies for concept acquisition; the latter describes a passive program such as Winston's, while the former refers to programs which use their current state of knowledge to decide upon suitable discriminating examples. Mitchell also discusses the idea of upper and lower bounds on a concept definition in slightly more general terms than above; rather than requiring that a prior hierarchy of features be completely known, he merely assumes that a partial ordering relation "is-more-specialized-than" can be defined upon such features. His discussion of the way in which instances and counterexamples of a concept can be used to bring these bounds closer together is very similar to that of Young, Plotkin and Linz.

After this consideration of some of the principal ideas in concept-formation in general, we now go on to consider the role which such ideas play in mathematical discovery, and look at some of the other mechanisms which are introduced in a program which operates within this domain, Lenat's program AM [Lenat1976a].

<u>AM: a proposed general model of mathematical discovery</u>

We shall now consider the most comprehensive and ambitious attempt to date to model the process of mathematical discovery, Lenat's program "AM", which is described in detail in his Ph.D. thesis[12], and is summarised in his "Computers and Thought" lecture at the fifth IJCAI ([Lenat1977a]). This program is a large and complex piece of work, and it will be necessary to examine closely its claims, its achievements, and its shortcomings. Other critical surveys of Lenat's work can be found in the paper by Hanna and Ritchie [Hanna1981a] and in the chapter on concept formation in Bundy's book on mathematical reasoning [Bundy1982b].

Briefly, the program begins with a body of knowledge about some domain chosen by the programmer (we shall discuss below the degree of domain-independence attained by Lenat), and uses a heuristic search technique to broaden its knowledge of that domain. When started with a knowledge of elementary mathematical concepts (Relation, Equality, Structure, Operation, etc.) and set-theoretic objects (Set, List, Bag, Set-union, etc.), AM develops concepts of number, arithmetic operations, and primeness, and proposes unique prime factori-

12 All page references, etc. in this section are to [Lenat1976a] except where otherwise stated.

sation and Goldbach's conjecture[13] among many other concepts and conjectures. Whenever a program displays a very high performance on a restricted collection of complex tasks, there are several questions which should be borne in mind while attempting to evaluate its achievement. In the case of AM we must consider the following:

- How sensitive was the precise choice of initial data? Was the quality of the result a consequence of a very carefully chosen starting configuration?

- How much were the program's heuristics "tuned" to produce the desired results?

- How well would AM adapt to other domains?

- How much further could AM have progressed if allowed to run for longer?

- Does the model possess any psychological validity[14]?

13 Alan Bundy [Bundy1982b] has pointed out that in the examples given in Lenat's thesis the conjecture called Goldbach's Conjecture is in fact a far more trivial conjecture, viz. that every even number is the sum of some number of primes (trivial because 4=2+2, 6=2+2+2, 8=2+2+2+2, etc.); however, the examples make it clear that AM could very easily have formed the correct conjecture by precisely analogous means - indeed it may well have done so on other runs.

14 This is neither a necessary property of an AI program, nor one claimed by Lenat for AM; it is, however a relevant question to ask of any program performing intelligent activities.

If not, does it at least give any insight into the structure imposed by people upon their knowledge of the world?

In order to attempt an answer to these questions, we shall now discuss in some detail the behaviour and internal structure of AM. We shall divide this into the following headings:

Passive Dynamic Knowledge ("Concepts")

Active Dynamic Knowledge ("Tasks")

Active Static Knowledge ("Heuristics").

The reasons for this choice of labels should become clear as the terms are explained.

Passive Dynamic Knowledge

By this heading we mean those parts of the program which are essentially treated as a declarative data-base by AM. Since AM is a production system, we can also describe it as the long term memory of the program. This data-base is constantly being modified and enlarged by AM, and indeed a large part of the measure of the program's achievement is the final state of the data-base. (One must also take into account the directness of the route by which this state was achieved).

The data-base is composed of a collection of "concepts", which are structures similar to frames [Minsky1975a], in that each concept possesses a set of "facets", each containing a particular type of information about the concept. It should be noticed that the set of possible facets is permanently fixed, and is the same for all concepts; there is no notion of a "type of concept" with a particular corresponding set of facets (nor is it clear whether such a notion is necessary or useful).

When a concept is first created (we shall discuss below how this can occur), many of its facets will be empty, or only partially filled in. In essence, the entire driving mechanism of AM is the attempt to fill in empty facets of known concepts. (This is similar to the control-structure of GUS [Bobrow1977a]).

Typical facets are: Names, Definitions, Specializations, Generalizations, In-Domain-Of (i.e. functions whose domain is the given concept), Worth, Analogies, Conjectures, Examples, Isas (i.e. concepts of which the given concept is an example). Also included as facets of a concept are "heuristics" (discussed in detail in the section after next) which tell the program how to fill in other facets of the concept, how to check existing entries for validity, how to estimate the concept's interest, and what activities pertinent to the concept might be worthwhile.

A typical concept might thus be (p.15):

```
NAME: Prime Numbers

DEFINITIONS:
    ORIGIN:
        Number-of-divisors-of(x)=2
    PREDICATE-CALCULUS:
        Prime(x)<=>(∀z)(z|x=>z=1 ∨ z=x)
    ITERATIVE:
        (for x>1): For i from 2 to √(x),¬(i|x)

EXAMPLES: 2, 3, 5, 7, 11, 13, 17
    BOUNDARY: 2, 3
    BOUNDARY-FAILURES: 0, 1
    FAILURES: 12

GENERALIZATIONS:
    Numbers, Numbers with even no. of divisors,
    Numbers with prime no. of divisors

SPECIALIZATIONS:
    Odd primes, Prime pairs, Prime uniquely-addables

CONJECTURES:
    Unique factorization, Goldbach's,
    Extremes of Number-of-divisors-of

INTUITIONS:
    "A metaphor to the effect that Primes are the
    building blocks of all numbers"

ANALOGIES:
    Maximally divisible numbers are converse
    extremes of Number-of-divisors-of;
    Factor a non-simple group into simple groups

INTEREST:
    Conjectures tying Primes to TIMES, to
    Divisors-of, to closely related operations

WORTH: 800
```

New concepts can be created in various ways as attempts are made to fill in facets; among the more obvious are the creations of generalizations or specializations. We shall defer full discussion of concept-formation to the section below on AM's "heuristics".

The particular facets "Examples" and "Isa's" relate together pairs of concepts in a lattice, as do the pair of facets "Specialisations" and "Generalisations"; the concepts are thus partially ordered by increasing specialisation, with the concept "Any-Concept" at the top of the hierarchy of concepts (there is an item "Anything" which lies above "Any-Concept" - this is the most general category known to AM). It is this lattice structure which Lenat describes as AM's "concept hierarchy".

Active Dynamic Knowledge

We discuss here the control structure adopted by AM for scheduling its activities. One of the possible effects of a heuristic is to create an object known as a "task". A task comprises: an activity to be carried out (e.g. "Fill in"); a concept and associated facet on which the task is to be carried out (e.g. "Examples of Number"); a value, indicating the worth of carrying out the task; and a list of reasons why the task was proposed. Tasks are arranged on an "agenda", which is a list ordered by the worth of the tasks.

The flow of control of AM is repeatedly to pick a task from the agenda, allot resources to it, and then carry it out until it terminates normally or exceeds whichever comes first of its allotted resources of either space or time. In the usual operation of the program the top task on the agenda (i.e. the one with highest worth) is always chosen. However, it is possible for the user to direct this choice interactively; also, Lenat carried out experiments in which the next task was chosen randomly from among the top twenty, or even randomly from the whole agenda - he reports that the first of these experiments led to a decrease in the "directedness" of the program's search, and about a threefold slowing in the rate of making "interesting" discoveries, whilst the second caused AM to thrash about vainly in a morass of expanding search-space.

Tasks are proposed, i.e. added to the agenda, as a result of various activities of the program. It is possible for the same task to be proposed several times; in such a case it is important that the worth of the task be raised only if it is being proposed for a different reason than before. This is the justification for the inclusion of symbolic reasons in tasks. In general, the worth of a task is computed from the ratings associated with the reasons supporting it, and the worths attached to the activity, the facet, and the concept involved. Lenat gives a rather complicated formula for this, originally intended as an ad hoc first approximation. He asserts that in fact the precise formula used is unimportant provided that it satisfies certain intuitively plausible monotonicity properties, and that the original formula proved satisfactory.

The programmer fixes the worths mentioned above associated with symbolic reasons, activities and facets; again, Lenat asserts that the precise values used are of little consequence.

Active Static Knowledge

We use this description for AM's "heuristics" because it is these heuristics which govern the actions carried out by the program, but the heuristics themselves are immutable.

Before going further, it is necessary to clear up a possible misunderstanding generated by Lenat's confusing terminology; in the view of AM as a heuristic search program, in the tradition of the Graph Traverser [Doran1966a], Lenat's "heuristics" do not correspond to the heuristics which control the search. Rather, they are the rules by which successor nodes in the search space are generated[15]. The search is in fact governed by the

---

15 This point is also made in Bundy's "rational reconstruction" of AM's search procedure [Bundy1982b], in which he represents some of the heuristics as inference rules, e.g.

$$\frac{\forall Ex\ (\mathrm{example}(C1,Ex) \Rightarrow \mathrm{example}(C2,\ Ex))}{\mathrm{conjecture}(C1,\ C1 \subseteq C2)}$$

which is to be read as

single heuristic "Use the worth of a task as an evaluation function; carry out the 'best' task"; thus, the nodes of the search space are tasks, and the generation of concepts and conjectures can be regarded as a side-effect of the search procedure. Having made this point, we shall from now on use Lenat's terminology without further comment.

AM's heuristics are production rules of the form

IF pre-conditions THEN action.

The pre-conditions are a set of tests on the current environment, and are constrained to have no side-effects on any of AM's data structures; typical tests would be "more than half the allotted space for the current task has been used", "concept C has no Examples", "the current task has found at least 10 entries for facet F of concept C", etc. Included amongst the tests there is always one of the form "the current task

---

IF all examples of C1 are Examples of C2
THEN add conjecture "C1 $\subseteq$ C2"
to the conjectures facet of C1

and others as "meta-level inference rules" to control the otherwise explosive search generated by these rules, e.g.

19. To fill in examples of X, where X is a kind of Y,
Inspect the examples of Y; some of them may be examples of X as well.
The further removed Y is from X, the less cost-effective this rule is.

is to perform activity A on facet F of concept C"; this test is in fact used to index the heuristics, as an aid to efficient retrieval of the heuristics relevant to a particular task.

The execution of a task involves gathering all the heuristics relevant to carrying it out (which in general involves "rippling" up the concept hierarchy to collect the heuristics associated with generalisations of the associated concept), and executing all those whose left-hand sides are satisfied, although this process may be affected by the restrictions imposed on the resources used by the task. To execute a heuristic, the action on the right-hand side is performed.

A right-hand side can in general do one or more of the following: suggest a new task, create a new concept, create an entry for a facet of an existing concept. When a new concept is created, certain of its facets are filled in at once, e.g. its definition and its name; in general only those things which are easy to fill in at creation time but would be harder in a subsequent task (because the present context provides relevant information) are filled in at once. New tasks will be proposed to fill in each of the empty facets of the new concept.

As an illustration of the creation of new concepts, Lenat gives the following example (p. 42)

Heuristic:
If the current task was (Fill-in examples of F),
and F is an operation from domain space A into range space B,
and more than 100 items are known examples of A (in the domain of F),
and more than 10 range items (in B) were found by applying F to these domain items,
and at least 1 of these range items is a distinguished member (especially extremum) of B
Then (for each such distinguished member 'b' in B) create the following new concept:

Name:	F-Inverse-of-B
Definition:	$\lambda(x)$ (F(x) is b)
Generalization:	A
Worth:	Average (Worth(A),Worth(F),Worth(B), ||Examples(B)||)
Interest:	Any conjecture involving both this concept and either F or Inverse(F)

In case the user asks, the reason for doing this is:
"Worthwhile investigating those A's which have an unusual F-value, namely, those whose F-value is b"
The total amount of time to spend right now on all of these new concepts is computed as:
Half the remaining cpu time in the current task's time quantum.
The total amount of space to spend right now on each of these new concepts is computed as:
The remaining space quantum for the current task.

We may note in passing that the entry on the Interest facet of the new concept seems to be the only form of new heuristic which is ever created by AM.

This heuristic was triggered while AM was working on the task "Fill-in examples of number-of Divisors-of", and created (among others) the new concept "Divisors-of-Inverse-of-Doubleton", defined by "$\lambda(x)$ (Divisors-of(x) is a Doubleton)"; (note that the "Definition" of a concept is in fact a predicate which is true if and only if its argument is an instance of the concept). Thus AM has defined the concept of prime number, and subsequently it goes on to explore this concept further, for example conjecturing that "the set of bags of primes whose product is x is always a singleton", i.e. the unique factorisation theorem. (See Appendix 5, tasks 149, 152, 178-181).

In addition to creating new concepts, heuristics can propose new tasks or fill in entries on a facet of an existing concept; rather than go into a detailed account of the ways in which this can happen, we shall consider one more illustrative example, in which a new conjecture gets proposed as a side-effect of the task "Check examples of Odd-primes" (p. 51). One of the relevant heuristics which is gathered for this task is (p.238):

```
56. If the current task is to Check Examples of
            concept X,
        and (Forsome Y) Y is a generalization of X
            with many examples,
        and all examples of Y (ignoring boundary
            cases) are also examples of X,
    Then conjecture that X is really no more
            specialized than Y,
        and Check the truth of this conjecture on
            the boundary examples of Y,
        and see whether Y might itself turn out to
            be no more specialized than one of its
            generalizations.
```

This heuristic was attached to the concept Any-Concept, and would thus be invoked for any "Check Examples ..."

task. When checking examples of odd primes, all examples of primes (ignoring the boundary cases) were found to be odd, and so an entry was added to the Examples facet of Conjectures: "All primes (other than '2') are odd primes". A new task was also proposed: "Check Examples of Primes", with the supporting reason "Just as Primes was no more general than Odd-primes, so Numbers may turn out to be no more general than Primes"; note that this task is a general one, in that all the heuristics relevant to "Check Examples of Primes" will be invoked, not merely the one relevant to determining whether all Numbers are Prime - thus the reason for proposing a task provides no guidance to AM on how to perform the task.

Strengths and shortcomings of AM

Having examined in some detail the working of AM, we are now in a position to consider its contribution to AI research, and the particular strengths and shortcomings which it exhibits. We can also attempt to answer some of the questions raised at the beginning of this section.

One of the strong points of the program is that its basic control structure is extremely simple; not only is the loop "select a task; collect heuristics; execute them" very straightforward, but the number of different kinds of tasks which the system can perform is very small (viz. four - Fill-in, Check, Suggest, Interest). However, as a corollary of the simple control structure, all the complex behaviour of the program has to be encoded in the heuristics and initial data - principally in the two hundred and fifty or so heuristics.

The only limitation on the power of the task agenda as a control mechanism is that the sphere of AM's activity must be amenable to representation as a structure of frame-like concepts, with a reasonably limited set of possible "slots" in the frames. Such a formalism seems general enough to cover many or most learning tasks. Although it might be arguable that in general one needs to be able to construct new types of facet, and there are certainly facets (e.g. Justifications, Counter-examples) which would need to be added to Lenat's set, it seems intuitively implausible (at least to the present author) that such slot-types can be multiplied indefinitely.

Thus, as observed above, essentially all AM's knowledge of how to carry out a specific activity, such as mathematical discovery, is contained in the heuristics. The question now arises: To what extent are AM's initial heuristics applicable to working with databases other than the "primitive mathematics" one used by AM? Lenat describes a "geometry world" experiment with AM; however, this world is structurally so similar to the original one that very little can be deduced from the experiment - in fact, beyond defining elementary concepts like congruence, AM seemed to spend much of its time rephrasing its number-theoretic work in terms of integer angles.

This is a symptom of an important distinction which Lenat does not draw in his work on AM, between abstractions and models. When AM has discovered "Bags-of-T" as interesting objects, it then goes on to explore their properties; this is interpreted by the user as the discovery of numbers. However, what is in fact being investigated is a particular model of numbers, and like other models it possesses irrelevant properties (e.g. each "number" is a sub-bag of many other "numbers"). If we were to define numbers actually to <u>be</u> "Bags-of-T", we might eventually discover some closely analogous objects (e.g. "Lists-of-nil" or nested sequences of sets) which had very many properties in common with "numbers" but were nonetheless quite different in other respects. At this stage a plausible possibility would be to define numbers purely intensionally, as the abstraction of the "interesting" common properties of "Bags-of-T", etc. - assuming that such properties could be determined. Of course, for such a definition to be useful one would require a system which had powerful tools for manipulating formal definitions, and this goes well beyond what Lenat has attempted in AM; we believe that one of the major limitations on AM's achievement is its need always to have "concrete" models to manipulate, since models of complex concepts are likely to be unwieldy, and many of their interesting properties may be more readily discovered by formal means than empirically.

Of course, many of the heuristics are specifically attached to relatively specialised parts of the domain, but many others are of very general application - almost half of the heuristics are attached to the very high-level concept "Any-concept". One might hope, then, that many of the heuristics are indeed appropriate for a wide variety of discovery tasks, and in fact a large number of them do appear to possess great generality (see Appendix 3).

However, careful study of the set of heuristics reveals a number of anomalies. Many heuristics seem to be at an excessively detailed level, containing information on how to decompose predicate calculus or recursive function definitions, or list-structure representations of objects. It seems that in his desire for structural uniformity, Lenat is in danger of confusing different levels of knowledge by according to what are essentially low-level manipulation routines the same logical status as is given to far more abstract rules of inference. Indeed, there may well be a case for replacing his single uniform rule-set with a multiple production rule system, i.e. a collection of rule-sets organised so that certain of them are available only in particular contexts (note that this should be distinguished from the indexing mechanism which Lenat uses to retrieve rules relevant to a particular task). It may also be remarked that many of Lenat's more general rules appear to be particular instances of even more general rule-schemas; a more economical, and cleaner, structure may be possible in which groups of syntactically and semantically similar rules are replaced by single meta-rules. For example, rules 47, 52 and 55 all essentially say "If (under various circumstances) a concept has few examples, try generalizing it", and the dual rules 48, 53 and 54 say "If a concept has too many examples, try specializing it"; these could perhaps be subsumed into a pair of rules, and possibly even into a single rule with a form something like: "Non-trivial concepts should possess reasonable numbers of examples and non-examples; a way of reducing/increasing the number of examples is by specializing/generalizing the concept". In a more sophisticated AM-like program, which was capable of generating new rules, we might expect to see the duality of specialisation and generalisation captured by a meta-rule which, given a rule involving one produced the dual rule using the other.

Of AM's 240 or so heuristics, about a quarter are principally concerned with directing AM's attention, especially with deciding which concepts are interesting and how interesting they are. These seem to fall into a different category from, for example, those heuristics which create new concepts; they correspond more closely to the "classical" form of heuristic search, in that they provide an "evaluation function" on concepts, which is in turn part of an evaluation function on tasks. This corresponds to the distinction observed by Bundy, as mentioned in an earlier footnote, between those heuristics which are essentially inference rules and those which are instead "meta-level" rules to guide the search.

There are slightly over 30 heuristics which explicitly construct new concepts; in addition to this, however, concepts can be created by the application of certain other concepts - e.g. Compose applied to two Active concepts yields another Active concept. There seems to be a certain taxonomical untidiness about a system in which the function of concept-formation is thus distributed between two quite different mechanisms, as also about a system in which the examples of some concepts are concepts, whilst those of others are not. This untidiness appears to stem at least in part from a lack of any clear distinction between _particular_ and _general_, a distinction which is indeed often hard to draw. (Is Add a particular instance of Operation, or is it a general class of triples (x, y, z) such that x+y=z? According to Lenat's taxonomy it is both). It is not at all clear what would be a proper remedy for this, and we shall do no more here than suggest that there is room for substantial re-thinking of AM's underlying ontology, and for re-organising the heuristic rules so that the genuinely "heuristic" ones are separated from the "rule of play" ones - and furthermore, so that both of these are separated from those which encode knowledge about the particular representations adopted by AM (e.g. the fact that sets are represented as LISP-lists sorted in lexicographic order).

We have criticised the "heuristics"; what of the choice of concepts in the original data base? Despite Lenat's claim that the initial set of concepts of the system corresponds approximately to those possessed by a child of about four (p. 113), the knowledge embodied in AM's starting state is articulated in ways much more formally sophisticated than would be implied by that claim. One important distinction which Lenat does not draw is that between possessing a concept at the level of being able to recognise instances of the concept as being members of a distinguished class with something in common (implicit possession of a concept), and possessing a concept explicitly, at the level of being able to introspect about the definition and structure of the concept. All of AM's concepts are of this second, explicit, kind; thus, it not only possesses concepts like Bag, Set, Ordered Set, and List, but knows clearly about the relationship and distinction between them, and possesses organising generalisations such as Ordered-structure, Structure-without-repeated-elements, etc. Thus, from the viewpoint of psychological validity, AM could be criticised for having its knowledge too well articulated.

However, as we noted in our opening remarks, Lenat makes no strong psychological claims for AM. The second part of our original question on psychological validity was "[Does the program] give any insight into the structure imposed by people upon their knowledge of the world?", and here the model of a hierarchical lattice of structured concepts acted upon by "heuristic" rules seems to be potentially very fruitful, and well worth further development.

A serious alternative to a "psychological" view of AM is to consider it as a logical system. According to this view, the program's significance lies in the empirical methods used to extend the initial set of definitions and assertions, and in whether such a system could go on extending itself indefinitely, or whether it must ultimately be overwhelmed by the "combinatorial explosion", as AM appears to be. Lenat claims that the eventual degradation of AM's performance is caused by the lack of new special-purpose heuristics to handle the new concepts defined; it is not clear that this is altogether the real reason, and indeed the converse can also be argued - that what AM lacked was a sufficiently powerful set of very _general_ focus-of-attention heuristics. In particular, a strong directing force which AM lacks is any sort of goal-driven activity; one would expect a much better performance from a program which could select interesting goals to work towards, although it is very much an open question how the relevance of candidate tasks to a goal might be estimated, and how the system could be kept from a dogged pursuit of one fata morgana after another.

Another apparent anomaly, we would suggest, is that one of AM's concepts enjoys a special status which is not made explicit anywhere in Lenat's thesis. This is the concept of equality, which is present explicitly as Object-Equality. Equality plays a fundamental role in AM's discovery of Number; furthermore, according to Lenat, if Object-equality is excised from the initial database it is not rediscovered by AM. However, many of AM's heuristics include checking objects for equality (in the sense of identity) without referring directly to this concept. Indeed, this seems perfectly reasonable, since it seems clear that the recognition of identity and difference does indeed play a fundamental role in any reasoning process; it is merely a little strange that Lenat nowhere discusses this special status, but merely assumes it implicitly.

We may note in passing a slight peculiarity relating to noticing equality: in task 29, p.297, "Check Examples of Set-Union", AM notices that "often Set-union (x, y) was equal to one of its arguments", and goes on to define the Superset concept, crucial to later development, as a result of this observation. We have been unable to find a heuristic in the list given in Lenat's appendix 3 which accounts for AM noticing this fact at all.

Related to this is AM's limited ability to notice analogies; in general, its knowledge about analogies stems from those which it explicitly constructs, and from heuristics which instruct it to look for similarities between concepts with a common generalisation. It possesses no mechanism for noticing totally "unexpected" analogies between totally unrelated concepts.

At the beginning of this section we posed the questions "how sensitive was the choice of initial data?" and "to what extent were the heuristics tuned?". These questions have also been raised by Hanna and Ritchie [Hanna1981a], who suggest that the data and heuristics were indeed designed to produce the particular behaviour shown by AM. However, in a reply to this paper [Lenat1981a] Lenat strongly rebuts this suggestion, and asserts that almost all of the concepts and attached heuristics were designed before AM was coded, and that virtually none of them was subsequently modified.

Furthermore, AM _failed_ to make some of the discoveries which its author had expected, and made a number of quite unexpected ones. Thus it appears that the answer to our questions is that both the rules and the initial concepts were _not_ specially chosen to produce a given performance.

Hanna and Ritchie also make a number of other detailed criticisms of Lenat's thesis; the essence of much of their criticism is that the thesis as it stands cannot be an accurate description of the program which produced the results described. Our own reading of the thesis would support this view to some extent, as is shown by the occasional detailed problem noted above; however, we would attribute the problem primarily to confusion engendered by Lenat's rather over-fanciful translations from LISP code to plain English. A question which remains unresolved is whether _all_ of the heuristics were explicitly represented by the program as separate rules, or whether some of the "rules" cited are in fact merely a commentary on behaviour which was coded into the program to embody a number of interleaved rules. Lenat, in the reply cited above, seems to concede that this is indeed the case, but later goes on to say that the control mechanism was precisely as described in the thesis, with no hidden subtleties. These two statements seem to be mutually incompatible, and there remains some confusion about this point.

A shortcoming which Lenat himself attributes to AM is its lack of any formal proof methods, or even the concept of proof. Whilst these would be of value in rejecting invalid conjectures, confirming others (and possibly thereby leading to new concepts), and perhaps in rejecting obviously futile tasks, the real benefit of such an addition would be the goal-directedness which it could give to AM. We shall discuss this in the next section.

The task of AM

Having considered the method adopted by Lenat for his chosen task, we must now consider briefly the task itself. At this stage, we are in some danger of criticising AM for not being what it lays no claim to be; these comments should be taken therefore less as a criticism of AM than as some ideas for further work arising from Lenat's.

The first, and most important, observation (already touched on above) is that AM models only a very limited form of discovery, namely discovery by data-driven (or forward) search. In practice much (we are tempted to say "almost all") mathematical discovery (and here we are tempted to replace "mathematical" by "scientific") is the result of goal-directed activity. That is not to say that the mathematician deliberately embarks upon the task of making a particular discovery, or even of making a discovery of a particular form. Rather, discoveries spring up as side-effects of trying to solve very specific problems; the discoveries may themselves be apparently remote from the problem being considered. One may instance here the considerable amount of mathematics which has arisen from the (unsuccessful) attempts to prove Fermat's Last Theorem [Edwards1977a].

Thus, an AM-like system should benefit from the incorporation of a problem-solving mechanism, and a component which selects tasks according to their apparent relevance to the problem at hand. The design of such a problem-solver would of course be a very large research project in itself.

The second observation is that AM searches for its discoveries within a formal domain. In many fields of science this is only the second stage of the discovery process, and not necessarily the harder. There must first come a stage of formalisation, developing the appropriate descriptive concepts and language[16] from empirical data. It is unclear how an AM-like system might go about this task. In reply, it might be argued that, at least in the particular domain chosen by Lenat, the initial concepts are to be regarded as corresponding

---

16 In reply to the argument that all knowledge is in some sense "formal", in that it is representable within some formalism, we would point out that some languages (e.g. English) are less amenable than others (e.g. lambda-calculus) to formal manipulation.

to "innate" knowledge, needing no prior concept-formation process. We find such an argument implausible if AM is to be regarded as having any psychological validity - there seems no reason to assume that any of AM's starting concepts are "innate" in humans; of course, in the view of AM as a purely formal system, the entire argument becomes irrelevant.

Finally, we shall observe that the discovery undertaken by AM is a single-level process. That is, AM can discover concepts, but not new discovery techniques; the set of heuristics is essentially inextensible. The remedy proposed by Lenat is a further "flattening" of the program's structure, so that heuristics themselves become instances of a Heuristic concept. Attractive though this uniformity may be, it seems to us important to keep a clear distinction between levels of abstraction. Thus, even though one may wish to keep a uniform representation for all kinds of objects known to the system, these should be collected in groups as "concepts", "rules about concepts", "rules about rules", and possibly further meta-levels. This remark may be considered in conjunction with our earlier suggestion that the heuristics may be better expressed as a multiple rule-set with rule schemas.

Paradigms for Deduction by Analogy

In this section we shall consider two ways of using analogy as a deductive tool. The first is proposed by Kling [Kling1971a], the second by Munyer [Munyer1977a]. Before describing these, we should like to quote Bledsoe [Bledsoe1977a] on the importance of analogy for deductive systems:

> "Perhaps the biggest error made by researchers in automatic theorem-proving has been in essentially ignoring the concept of analogy in proof discovery. It is the very heart of most mathematical activity and yet only Kling (1971) has used it in an automatic prover. His paper showed how, with the use of knowledge, a proof in group theory would be used to help obtain a similar proof in ring theory.
>
> "We strongly urge that other workers in this field familiarize themselves with Kling's work and extend and apply it more effectively."

The work we shall describe by Munyer may be seen as an attempt to follow the advice in Bledsoe's second paragraph. Before discussing it, we must examine Kling's contribution towards the understanding of the use of analogy as a deductive tool.

Kling's program ZORBA

Kling's fundamental idea is extremely simple: many resolution proofs are rendered intractably large because a very large search space is generated by the presence in the initial database of a large number of clauses irrelevant to that particular proof. If the initial clause-set can somehow be filtered to include only those axioms which will directly contribute towards a proof, then the resulting much smaller search space is far more likely to lead to a proof being found.

The means adopted for filtering the database is as follows:

> Given some theorem T to be proved, and an already proved theorem T' together with its proof P', an analogy mapping A from T' to T is constructed. This mapping is applied to the set of clauses used in P', and the resulting set of clauses used as a database for attempting to prove T.

This, it is hoped, leads to a very substantial reduction in the search space, and renders feasible a previously impossibly explosive proof.

Kling also suggests an extension of this algorithm, where the lemmas used in proving T' are mapped into corresponding lemmas for proving T; it is of course no longer necessarily the case that the generated lemmas are true, nor that they are relevant, but at least it seems plausible that some of them will contribute effectively towards a proof.

Kling suggests that the lemmas be solved before attempting a proof of T; this is neither necessary nor obviously better than the alternative "lazy evaluation" strategy of deferring their proof until it is known to be needed. There is yet a further step, which Kling does not take, that seems to follow immediately from the previous ideas: one could take, in addition to the analogues of the lemmas used in the proof of T', the analogues of the clauses used in the proof, and if they were not already clauses in the database (as is required by the first algorithm described) treat them as lemmas to be used subject to verification of their validity; this is one of the bases of Munyer's approach.

It should be noted that Kling's paradigm discards a great deal of useful information from the original theorem and proof; no attempt is made to use any information about the order in which clauses were used, nor which literals were resolved upon. To express the same point in a wider context, the proof P' may well be closely structurally analogous to some proof P of T (as is indeed the case with Kling's examples from abstract algebra); the above method discards almost the entire structure of P', and repeats the entire search[17].

An extreme alternative to this method would be to

---

17 Rather as though, possessing a recipe for lamb casserole, and wishing to cook a beef stew, we noted that we were likely to need beef, onions, potatoes, carrots, stock, salt, an oven, a knife, a dish and a work-surface, and then threw away the recipe book without reading the method of preparation.

take the entire analogue of P′ as a "proof plan" for T, attempting to justify each step in turn; one could then envisage the entire process as a recursive one, each step of the proof plan being worked on by the analogy mechanism. This is indeed extremely close to what Munyer does.

A few further points should be noted about Kling's work before we move on to consider Munyer. In Kling's program ZORBA, it is the user who selects the analogous theorem T′ and supplies its proof P′. Thus ZORBA consists essentially only of the analogy-formation mechanism, plus a resolution theorem-prover (QA3 [Green1969a]). The analogy mechanism is used repeatedly in the attempt to prove a theorem, constructing ever larger initial databases using ever laxer analogies until a proof is successfully found. Kling's description of his algorithm for constructing analogies is very detailed, but lacks any clear overall summary; it appears essentially similar to the technique used by R.Brown [Brown1977a].

The user also supplies ZORBA with a set of "semantic templates", which provide type information about the functions and predicates used in the database; these templates are used to reduce the search for possible analogies by ensuring that argument-types are mapped consistently[18].

---

18 R.Brown [Brown1977a] points out that it is in general possible to determine these semantic templates automatically by a simple syntactic criterion based on the structure of the assertions which contain the

The important point to note here is that the possible analogies are being restricted by semantic considerations; in this respect Kling has more to offer than Munyer, whose work we shall now examine.

Analogy Viewed as a Cousin of Unification

Munyer's philosophy can be summed up by two quotations from his later paper [Munyer1977a]:

> "Although the solution to a theorem-proving problem must be logically rigorous, the means by which it is discovered need not be."

> "How to use an analogy turns out to be at least as important as how to find an analogy".

His proposed system follows both of these maxims, in that it makes steps which are not necessarily logically valid in its formation of proof plans, and in that the method by which an analogy is actually sought is an extremely naive exhaustive search.

His approach resembles that of F.Brown [Brown1977b], or of STRIPS [Fikes1972a], insofar as his proposed system is an extensible deduction system, in which previously proved theorems are assimilated into the system and are used to contribute to further proofs. The way in which this is done is related closely to the STRIPS approach of using "MACROPS", since each proof known to the system (or

---

predicates in question. We consider his work below.

any subsequence of it) is available as an operator which can be applied to an intermediate goal (using the term loosely) in a proof to generate a sequence of subgoals. The principal novel feature of Munyer's method is that the applicability of an operator is determined by an analogy match between the goal and the operator.

The objective which Munyer's system seeks to achieve is to generate by analogy a proof plan for some theorem, in the form of a linear sequence of subgoals each of which can easily be verified by a simple conventional theorem-prover or proof-checker. The number of steps in a valid deduction of each subgoal from its predecessor should be very small, so that little or no search is done in going from the plan to a proof.

Operators are of the form

T1 => T2

where T1 and T2 are predicate calculus terms. Associated with each operator are: an analogy match B between subterms in T1 and subterms in T2 (not in general either injective or surjective), and a "degree of certainty" (DOC), representing a heuristic estimate of the plausibility of the derivation of T2 from T1 (DOC is a number between 0 and 1, and is 1 whenever (T1=>T2) is known to be a logically valid deduction). An operator can be applied either forwards or backwards, that is, either by matching

T2 against some goal-state T2' in a partial proof plan, or by matching T1 against some start-state[19] T1'. These two cases are precisely symmetrical; we shall describe the latter.

Suppose some analogy match A has been found between T1 and T1'; associated with this there will be a DOC reflecting the closeness of the match, which will be 1 when the match is a valid unification. Then we wish to use the maps A and B to generate a new subgoal T2' such that T2' is to T1' as T2 is to T1; this is the "classical" analogy problem as dealt with by Evans [Evans1967a]. We can represent the various formulae and mappings as follows:

```
T1 <-- A --> T1'
 |
 |B
 |
\ /
T2 <-- A'--> T2'
```

Difficulty arises when, as is frequently the case, no such T2' exists; it is then necessary to construct a "best guess". In any case, once a T2' has been found the step (T1'=>T2') can be added to the proof plan; associated with it will be a degree of certainty derived from the reliability of the analogies A and B, and the DOC of the operator (T1=>T2), together with the likelihood that (T1'=>T2')

---

19 "Start-states" are derived by forward reasoning from the preconditions, "goal-states" by backward reasoning from the conclusion .

will be part of an eventual solution path. Thus, analogy is being used as a sort of "fuzzy unification" to match terms in a form of *modus ponens* reasoning.

What we have just described is a "blind step" in the search for a proof: we have determined that an operator is applicable and applied it; a more desirable circumstance is that T1 and T2 simultaneously match via the same analogy to a start state T1' and a goal state T2', both of which are already part of an existing plan, thus making it more likely that the operator will be relevant to an eventual proof. When this occurs, the analogues of the intermediate steps of the (perhaps fuzzy) deduction of T2 from T1 can be directly mapped into a sequence of intermediate subgoals to be added to the proof plan. However, it is often the case that the analogies A between T1 and T1' and A' between T2 and T2' will be different; in this case, what Munyer calls a "skewed" plan is generated. We shall consider how to cope with skewed plans in due course, after considering the sort of analogy which Munyer's matcher will produce, and the ways of constructing a T2'.

The analogy matches correspond for the most part to second-order unifications or generalisations; for example, identical terms match against each other (with a DOC of 1), as do any pair of first-order unifiable terms. A pair of terms such as $\langle f(a),f(b)\rangle$ match fuzzily, as do pairs such as $\langle f(a),g(a)\rangle$, $\langle f(a,b),f(b,a)\rangle$, and $\langle f(a,b),f(b)\rangle$.

The diagram below shows how corresponding symbols in pairs of terms may be mapped in a few instances.

```
f (a)      f (a)      f (a)     f (a, b)     f (a, b)
| |        | |        | |       |   \/       |    /
f (a)      f (b)      g (a)     |   /\       f   (b)
                                f (b, a)
```

In cases such as (f(a,b) <--> f(b,a)) the DOC of the match will depend upon whether f is known to be commutative[20]. We may note that the matcher will always find some match between any two terms, and that it is not guaranteed always to find a valid second-order unification, even when one exists.

The next matter to be considered is the generation of a term T2' from T1', T1, T2, A and B. In describing how this is done, Munyer has made some apparently arbitrary choices; he does not discuss the reasons for these particular choices, and the only evident justification is the empirical observation that they work for the problems he has considered so far. We shall give a few examples of the construction of T2', paraphrasing [Munyer1977a, p5].

> "If a symbol in T1' is mapped to a symbol in T1 which is in turn mapped without change to a symbol in T2, then the symbol from T1' appears in T2' (e.g. example 1). If however the symbol in T1 is mapped to a different symbol in T2, then the symbol from T2 appears in T2' (example 2). In either case, if the corresponding symbols in T1' and T1 are different,

---

20 An interesting problem would be the automatic generation of lemmas such as the commutativity of some function which was frequently used in such matches.

the DOC of the step is lowered[21].

"A permutation among the arguments of a function in going from T1 to T2 is copied among the symbols in T1' to which they are mapped (example 3). A permutation in going from T1' to T1 does not affect the formula which is produced but it does lower the DOC unless the containing function symbol in either T1' or T1 is marked as commutative.

"A symbol in T1 which does not map to a symbol in T1' does not appear in T2'[22] but the DOC is lowered unless an appropriate attribute[23] is present (example 4). A symbol in T1' which does not map to a symbol in T1 is considered to be unaffected by the operator and appears in T2', but the DOC will be lowered unless an appropriate attribute is present (example 5)."

	T1	T2	T1'	T2'
Example 1:	f(a)	f(a)	f(b)	f(b)
Example 2:	f(a)	f(c)	f(b)	f(c)
Example 3:	g(b,a,c)	g(a,c,b)	f(a,b)	f(b,a)
Example 4:	f(g(a))	f(g(b))	f(a)	f(b)
Example 5:	f(a,b)	f(b,a)	f(a,g(b))	f(g(b),a)

We can now consider how the system would go about

---

21 This decision appears arbitrary; it is not obvious that it would not be as good to copy the symbol from T1' rather than from T2, in which case example 2 would be replaced by
T1=f(a); T2=f(c); T1'=f(b); T2'=f(b)

22 This seems reasonable for his example 4, but consider
T1=f(a,b,c); T2=g(a,b,c); T1'=f(b,a);
it is not obvious that T2' should be g(b,a), as Munyer's rule implies, rather than g(b,a,c).

23 An attribute is some feature such as commutativity, associativity, etc. which may be associated with a function symbol to indicate that certain kinds of match are exact, not "fuzzy".

seeking a proof of a theorem. The sequence of actions performed is as follows:

A step to be worked on is chosen by a heuristic merit rating (which in general prefers the verification of plans to the taking of blind steps).

If a blind step is to be taken, the appropriate formula (T1' or T2' according as the step is backward or forward) is generated and added to a search lattice. If a plan is to be verified, the step with smallest DOC[24] is found, and its start and goal added to the lattice; the plan is then ineligible for further consideration until this step has been verified[25].

Whenever a new formula F is added to the lattice, it is first checked for subsumption or identity with all other formulae already in the lattice, and any subsumptions found are marked appropriately (to avoid carrying out essentially the same task several times).

Next, all analogies between F and theorems in the database are found. For each (sufficiently good, one

---

24 on the grounds if this fails there is no point wasting effort on the rest of the plan

25 Another possibility would be to re-activate the plan as soon as the considered step had achieved a high enough DOC to be no longer the weakest link.

presumes, though Munyer does not say so explicitly) analogy found, the theorem is searched for a second analogy which can be used to form a plan (i.e. steps of the theorem are matched against formulae in the lattice). For each such plan, it is corrected if skewed (see below), otherwise an appropriate inference link is added to the lattice, using the plan as an operator; if the DOC is not 1, the plan itself is marked as a candidate for future verification. If no plan is found for this analogy, it is instead used to propose one forward and one backward blind step.

For each added inference link, adjust DOCs appropriately; if the link completes a plan step, re-activate the corresponding plan.

Repeat the cycle until a solution is reached.

We have mentioned "skewed" plans several times. These occur when an operator T1=>T2 matches against formulae T1' and T2' by different analogies, so that a blind forward step from T1' would result in T2" (different from T2'), whereas a blind backward step from T2' would result in T1" (different from T1'). Munyer's brief explanation of how this is patched up is very sketchy, and his chosen example unilluminating; however, what he appears to be proposing is that consideration is given to replacing either T2' by T2" or T1' by T1" in the search lattice as "plan-correcting step".

The most important observation to make about this entire mechanism is that there is a very serious problem of controlling search. The mechanism is proposed as being itself a powerful tool for reducing the search space when seeking a proof:

> "It appears, based on this hand simulation, that the construction of the solution would be optimal in that no search (blind steps) is required and no incorrect steps are actually generated." ([Munyer1977a], pp. 9-10).

However, this claim needs justification which Munyer does not offer; indeed the rather crude method used for seeking analogies is liable to become disastrously explosive as the database of theorems grows. Thus Munyer is replacing one search problem by another, and proposing no solution to this second problem. Once again, we are presented with the fundamental problem of how to recognise an analogy amongst a large body of existing knowledge. A question which Munyer does go far towards answering is that of how such an analogy might be used once it has been found; this accords precisely with his already quoted remark that "how to use an analogy turns out to be at least as important as how to find an analogy".

<u>Brown's Work on Reasoning by Analogy</u>

In this section we consider the work of R. Brown [Brown1977a,Brown1976a] on the use of analogy mappings to transfer procedural "expertise" from one domain to another.

Although this is not an appropriate place to go into a detailed critique of the relative merits of procedural and declarative representations of knowledge, it is necessary to observe, before proceeding further with discussion of Brown's work, that an issue of debate in AI has been whether knowledge is better represented "passively" by declarative descriptions, or "actively" by procedures which embody the application of that knowledge (or, if both are appropriate, which is better in given circumstances). A more recent development has been the view that there is, in fact, little or no essential difference between these forms of representation; it is hard to generalise fairly, but one could perhaps say that the majority of those who would still claim that there is a significant difference between procedural and declarative representation fall into the procedural camp.

Brown's model of expertise consists of three tiers:

(1) <u>Code</u>: the programs which are actually run in order to carry out tasks in the domain world; these programs are low-level and detailed, and contain information

governing flow of control;

(2) Plans: these are essentially program outlines without any control flow information; they "specify goals, intentions and constraints";

(3) Descriptions: declarative assertions about the world, i.e. a set of definitions and axioms in a predicate-calculus-like language.

The immediate impression made by Brown's examples of these three levels of his world model is that the objects at all three levels are, in fact, executable programs, written in successively higher level languages. Thus his code examples are imperative programs in LISP, complete with the full armoury of PROGs, GOs and SETQs to demonstrate that they are real live Programs in all their naked horror. His plans are essentially sequences of pattern-matching manipulations on the representations of objects in the model worlds, and as such bear a very close resemblance to programs in some cousin of PLANNER [Hewitt1969a]. His assertional descriptions are more or less predicate calculus clauses translated into LISP notation, and would thus be regarded by many as executable programs in a logic interpreter similar to Prolog [Clocksin1981a].

Thus, by adopting his multi-level view of knowledge, Brown weds himself firmly to the "there *is* a difference!"

side of the procedural/declarative controversy-controversy, and comes down on the side of a low-level procedural representation of expertise; in taking this position, and in much of his subsequent development of the analogy mechanism, Brown's approach shows a close affinity with Sussman's in his program HACKER [Sussman1973a].

To summarise very briefly the very detailed technical description of Brown's analogy mechanism in [Brown1976a], analogies are constructed between some already known area of expertise (the _domain_) and some new area (the _image_) as follows:

(1) Use the assertional descriptions to propose a mapping between domain names and image names; this process is essentially syntactic, although Brown uses "semantic" type constraints on, for example, mappings of functions and predicates (an extension of Kling's use of "semantic templates" [Kling1971a]).

(2) Use this map to translate plans and code in the domain world to plans and code in the image.

(3) Use plan-justifications to prove the translated plans correct; if this fails, use the justifications and descriptions to debug the plans. Similarly, verify and debug the translated code.

The debugging process appears similar to HACKER's.

It is clear that very sophisticated matching is required in order to determine which image-world assertion is sufficiently like which domain-world assertion to account for a "bug", and enable it to be fixed; indeed, such a matching would seem to constitute a large part of a general solution to the problem of producing analogies purely by inspecting descriptions.

This last point leads to the observation that there is a strong case for arguing that the entire analogy process should indeed be carried out at a descriptive level. That is not to say that predicate calculus without control information is necessarily a sufficient language to describe all domains of expertise; but languages like PROLOG have shown that it is possible to write programs with a declarative semantics, where the control structure is provided by the "machine" in which the program executes, rather than being an inherent property of the language.

It is apparent in Brown's work that most of the complexity arises from his multi-level representations, since he has to construct a whole sequence of maps between domains, and between levels within a domain, and then use compositions of these maps to construct hypothesised new pieces of "expertise", which still remain to be debugged. Whilst we would not wish to imply that the problem of constructing and using analogies is anything other than extremely difficult, it does seem that Brown's choice of knowledge representation formalism creates a great deal of added complexity without demonstrably providing greater expressive power than simpler options.

We shall now look in rather more detail at the most interesting aspect of Brown's work: the construction of the analogy map between his descriptive assertions.

The construction of an Analogy Map

Brown's maps are constructed at the level of his descriptions. The first stage of constructing a map is the discovery of semantic templates, similar to Kling's; these are automatically extracted from the descriptions by using the observation that type-checking predicates are unary predicates which appear quantified on the left-hand sides of implications. Consider, for example, the descriptions below (taken from [Brown1976a]). Brown's LISP notation has been changed to that of predicate calculus.

```
/** PLANE GEOMETRY DESCRIPTIONS **/

∀(A,B) [ pt(A) & pt(B) =>
             ln(line(A,B)) & in_ln(line(A,B),A)
             & in_ln(line(A,B),B) ]
   /* There is a line containing any two given
      points */

∀(A,B) [ distinct(A,B) & pt(A) & pt(B) =>
     ¬ (∃(X,Y) (distinct(X,Y) & ln(X) & ln(Y)
          & in_ln(X,A) & in_ln(Y,A) & in_ln(X,B)
          & in_ln(Y,B))) ]
     /* There is at most one line containing two given
        distinct points */

∀(A,B,C) [ pt(A) & pt(B) & pt(C) & between(A,B,C) =>
     ∃(L) (ln(L) & in_ln(L,A) & in_ln(L,B)
          & in_ln(L,C)) ]
     /* If B is between A and C then A, B, C are
        collinear */

∀(A,B) [ ln(A) & ln(B) & distinct(A,B) =>
     in_ln(A, intersect(A,B)) & in_ln(B, intersect(A,B)) ]
     /* The intersection of two lines lies in each of
        them */
```

The above rules form part of an axiom system for plane geometry, and it can be seen that the unary predicates which appear on left-hand sides are pt and ln, which are thus assumed by the analogy algorithm to be type-checking predicates.

The semantic templates which can then be constructed are:

```
in_ln(ln,pt)
line(pt,pt)
```

i.e., the arguments of in_ln must be of type ln and pt respectively, and those of line must be of type pt. As Brown observes, this extraction of semantic templates is in fact a purely syntactic procedure.

Suppose we wish to construct an analogy map from this domain to the domain of solid geometry, which will include descriptions such as:

```
/** AXIOMS FOR SOLID GEOMETRY **/

/* The first few axioms are identical with those for
   plane geometry */

∀(A,B,C) [ pt(A) & pt(B) & pt(C) &
              non_collinear(A,B,C) =>
              pl(plane(A,B,C)) & in_pl(plane(A,B,C),A)
              & in_pl(plane(A,B,C),B) &
              in_pl(plane(A,B,C),C) ]
     /* There exists a plane containing 3 given
        non-collinear points */

∀(P) [ pl(P) => ∃(A) pt(A) & in_pl(P,A) ]
     /* Every plane contains a point */
```

The requirement for an analogy mapping is that once a mapping has been defined for the type-checking predicates, it should be extended to the rest of the symbols in such a way that argument-types are mapped consistently. In the example of plane and solid geometry, the initial mapping is done by a heuristic which tries to map types of the same name in different domains to one another; this is obviously open to the criticism that the choice of names for predicates is a "secret" way of giving advice to the program (cf. the comments in [Hanna1981a] on Lenat's use in AM of the rule "If the user has recently renamed this concept then it becomes more interesting"). Thus in the above example, the mapping

```
ln -> ln
pt -> pt
```

would be chosen by this heuristic[26]

Subsequently, the *formation* of a consistent map can be viewed as a filtering problem for labellings of a graph, and as such can be handled by algorithms similar to that of Waltz [Waltz1975a]; it should be noted that in general Waltz's algorithm itself is not sufficient, since the analogy map requires global consistency of the labelling, whereas Waltz's algorithm only ensures local consistency. (An extensive discussion of such algorithms is given in [Freuder1978a], while a discussion of the different possible kinds of inconsistency in a graph labelling is given in [Mackworth1977a]).

Using the Analogy Map

If the entire process of constructing an analogy were as described above there would be little more to be said; however, the problem arises that such a mapping between the symbols in two domains is unlikely to be an exact analogy, in the sense that true statements and correct algorithms in one domain will not necessarily map to corresponding true statements and correct algorithms in the other. For example, the image of a theorem proof in

---

26 It is interesting to note that without this heuristic, the mapping (ln->pt,pt->ln) would be investigated; this is of course the first step of inventing projective geometry, as Brown observes in [Brown1976a].

plane geometry may well not be a rigorous proof on solid geometry, but rather a sequence of lemmas which may constitute an outline proof, needing completion and possible correction.

It is this need to "debug" inexact analogies which leads Brown to his rather baroque system of knowledge representation. He considers that the ultimate goal of an analogy is to aid the transfer of expertise from one domain to another in the form of <u>programs</u>. He achieves this by using the map constructed at the level of <u>descriptions</u> to map <u>plans</u> between domains, and using the images of the plans to construct programs. To ensure the correctness of images of plans, he uses <u>plan justifications</u>, which are proofs of plan correctness in terms of the axiomatic descriptions and definitions. Furthermore, he requires <u>commentary</u> attached to programs to show how they relate to plans.

It is unclear why the ramifications of such a representation should stop at this point, rather than requiring, for example, "plan justification commentaries" to show how a plan justification corresponds to a plan, etc. Conversely, even if we accept Brown's implicit belief in the need for a procedural representation fundamentally from the declarative one, it is not clear why his <u>plans</u> are not acceptable as such a representation, so that the goal of the analogy system becomes the transfer of correct plans, without the added layer of complexity afforded by programs.

It is not, then, surprising that the mechanism which Brown requires to carry out his many-layered mapping and debugging process between two of his domains is both cumbersome and confusing. While it may well be that such a degree of complexity is indeed required of an analogy system, this is by no means justified by the relatively simple instances given by Brown; his insistence on a low-level procedural representation of knowledge serves more to obfuscate the process of constructing and using analogy than to provide a clear explanatory model.

The Use of Analogy in Knowledge Representation

We have gradually moved away from our first focus of attention, mathematical discovery, towards a consideration of the use of analogy in general. The remaining sections of this survey will consider a number of approaches to the use of analogy in reasoning and knowledge representation. We begin with a discussion of the design proposed by Moore and Newell for a system whose entire representation formalism is based upon analogy, Merlin.

Can Merlin Understand?

In their paper "How Can Merlin Understand?" [Moore1973a] Moore and Newell describe a proposed formalism for knowledge representation which is pertinent to the present discussion on analogy. According to their formalism, all concepts known to the system are potentially "viewable as" instances of other concepts, subject to a suitable mapping being made between the components of the two concepts. This is precisely the main goal of an analogy-finder (the subsidiary goal being to evaluate the strength of the analogy once found).

Moore and Newell claim that their formalism is embedded within a system which "understands", and cite the following criterion for use of the word "understand":

> [A subject] S understands knowledge K if S uses K whenever appropriate.

Applying this criterion to the question posed by the title of their paper, the answer appears to be that Merlin _cannot_ understand at all, since Merlin (as they describe it) is simply an embodiment of their knowledge representation formalism, and of rules for reorganising its knowledge in response to requests to do so, or in the course of assimilating new knowledge. Thus, although Merlin might conceivably serve as the underlying basis for an understanding system, any such system would require as a major further part an _active_ component which would make use of Merlin's data-structures. Such a component would serve to provide an _interpretation_ of Merlin's knowledge, without which Merlin cannot be said to "use" its knowledge at all, appropriately or otherwise; as we shall see, Merlin itself provides no such interpretation.

However, the main point of interest here is what Merlin _can_ do, which is to construct analogies and to assimilate new data by analogy. Indeed, the entire knowledge base can be regarded as being organised by analogy, and in many ways the view of knowledge representation embodied in the program corresponds closely to the present author's.

The fundamental building-block in Merlin is an object called by the authors a "β-structure" (chosen as a neutral name which leads to no preconceptions about its interpretation). A β-structure is denoted

$$\alpha\text{: } [\beta\ \alpha 1\ \alpha 2\ \ldots\ ]$$

read as "α is a β further specified by $\alpha 1$, $\alpha 2$, ...". The components β, $\alpha 1$, $\alpha 2$, etc. are themselves β-structures.

An interpretation of a β-structure is "α can be viewed as a β given that $\alpha 1$, $\alpha 2$, ..."; this interpretation corresponds to a datum α being assimilated to a known datum β, where the αi can be viewed as defining an analogy between α and β. A map from β-structure B1 to B2 is notated B1/B2, and corresponds to a way of viewing B2 as further specification of B1.

As an example, consider the following, given by Moore and Newell:

Suppose we have

MAN: [MAMMAL NOSE:[...] HOME:[...]]

PIG: [MAMMAL SNOUT:[...] STY:[...]]

and wish to find an analogy between MAN and PIG (view a PIG as a MAN). The result will be

PIG: [MAN SNOUT/NOSE STY/HOME]

assuming that the maps SNOUT/NOSE and STY/HOME can be constructed; the interpretation would be "a PIG can be viewed as a MAN if his SNOUT can be viewed as a NOSE and his STY as a HOME".

For a full explanation of this example, and further examples, the reader should refer to the original paper [Moore1973a]. There are two main difficulties with the approach taken by Moore and Newell, one practical and one philosophical. The philosophical problem is that, since there are no "primitive" β-structures, the whole knowledge edifice seems to be built on air[27]. In this, Merlin's knowledge-base is similar to that produced by Quillian [Quillian1968a] in his "Semantic Memory" system. Whether this is truly a problem depends upon one's point of view; on the one hand those with a foundation in mathematical logic and related disciplines are likely to be horrified at the idea of such a "baseless" system, while on the other hand there is a strong intuitive appeal (for some) in the notion of a system where every definition can be further refined in terms of other definitions as far as necessary in any particular circumstances.

The final remark leads us to the practical problem: when does the recursive sequence of matching stop? This is a point on which Moore and Newell are most unclear; it is closely related to the question: under what circumstances can an attempt to view X as Y fail? (since obviously a failure to match corresponding components of a β-structure would cause at least that branch of the recur-

---

27 Or perhaps supported on "turtles all the way down"?

sive matching process to terminate). Again, the answer is not readily to be drawn from the paper.

Algebraic Models of Analogy

Two recent papers [Farreny1982a] and [Potschke1982a], propose an algebraic model of analogy formation, in which an analogy is represented as a homomorphism between algebras or (equivalently) between graphs.

Both papers represent the situations between which analogies are to be constructed as relational algebras. That is, a situation is described as a set of objects together with a collection of relations defined upon that set. For example ([Farreny1982a]), the situation

```
Romeo loves Juliet.
Juliet loves Romeo.
Romeo is a man. He is Italian.
Juliet is a woman. She is beautiful. She is unmarried.
```

consists of the set

```
{ Romeo, Juliet }
```

and the relations

```
loves =      { (Romeo, Juliet), (Juliet, Romeo) }
Italian =    { (Romeo) }
man =        { (Romeo) }
woman =      { (Juliet) }
beautiful =  { (Juliet) }
unmarried =  { (Juliet) }
```

where a relation is represented as a set of tuples from the underlying set.

An analogy between two situations is now defined to be a mapping between the corresponding objects which preserves (or nearly preserves) relations. In Pötschke's paper, he first defines an analogy to be a strict homomorphism between algebras, but then points out that this is not always guaranteed to exist, and goes on to mention briefly the idea of a "loose" analogy constructed from an approximate homomorphism. He indicates a possible measure of the closeness of such a mapping using the ideas of positive defect and negative defect of a mapping between labelled directed graphs - the number of edges which need to be added to the domain or deleted from the range, respectively, such that the mapping is a homomorphism. However, he does not give any indication of how such approximate mappings may be found; nor is it clear how he would represent a general situation, which may contain relations more complex than binary ones, as a labelled directed graph.

The second half of his paper gives an algorithm for carrying out analogy-formation in the style of Evans [Evans1967a] given three graphs A, A' and B, and a map A→A'. This involves the steps "Compute a maximal common partial graph of A and B" and "Generate a minimal set of substitutions S={S1,...Sk} such that S(A)=S1(S2(...(Sk(A)...))=B". Both of these steps are liable to be computationally expensive, and he does not suggest algorithms for them. It should also be noted that "maximal", in the mathematical sense, does not mean "largest possible", but rather "not enlargeable"; there may be many maximal common partial graphs of A and B, and he does not discuss the criteria for choosing between them.

The examples which he gives are small; there is no indication of how effective his methods would be in constructing analogies between complex situations. However, his use of positive and negative defects in measuring the looseness of an analogy may provide a possible "dissimilarity metric" between concepts, in the sense discussed below.

Farreny and Prade discuss at some length the possibility of using "semantic similarity" as a criterion for mapping one relation to another; they base their ideas on the notion of "fuzzy sets", as discussed by Zadeh [Zadeh1979a]. They assume that properties to be matched by analogy denote "fuzzy" classes with associated probability measures of the likelihood of a datum possessing the property. The degree of similarity between two properties is then defined as the likelihood of a datum belonging to both classes. As the authors themselves admit, this is a far from general model. Whereas for adjectives such as "tall", "short", "old", "young", etc. it is clear what the appropriate universe of discourse is, and it seems apt to use possibility measures in such cases, there are obviously many cases where this is not so. In general, such a model is only appropriate where the properties describe subsets of some quantitatively measurable overall attribute (e.g. "height", "age"). Although they refer to a need for further work in the area of measuring "semantic similarity", Farreny and Prade do not themselves go into detailed consideration of the possibilities.

The construction of a map between situations is presented as a problem of matching labelled graphs, as in the work of Pötschke and R.Brown discused above; there is no consideration of the details of an algorithm, but clearly the authors' intention is to map together semantically similar properties; the degree of similarity would then provide a measure of the closeness of the analogy. This approach seems to neglect the view that often the most valuable analogies are those between apparently dissimilar concepts.

Analogy by Means-Ends Analysis

We conclude this survey with an examination of Carbonell's work on problem solving by analogy [Carbonell1981a]. Carbonell holds the view that such problem solving is very closely linked to learning by experience; previous knowledge is so structured as to be retrievable through similarity to new problem situations, and the success of plans developed by analogy can lead to fruitful generalisation of prior knowledge.

His proposed problem solving strategy is an extension of classical Means-End Analysis[28] as considered in depth by Newell and Simon [Newell1972a]; a problem state is reduced to a solution state, or goal state, by the successive application of operators which reduce the difference between the two states. An attempt to use previously known solutions as a means of reducing search in MEA was made in the program STRIPS [Fikes1972a], which stored all subsequences of previous solutions as compound operators ("MACROPS"); as Carbonell points out, the search amongst applicable operators then becomes rapidly computationally infeasible, as the number of operators increases; thus STRIPS can be seen as replacing one form of search by another, with no clear evidence that the latter is ultimately more efficient (this is the same criticism which we

---

28 Henceforward "MEA"

earlier applied to Munyer's work).

Carbonell proposes a _reminding_ process to compare the initial and final states of, and path constraints[29] on, a new problem with those of previously solved ones, and to compare the applicability of operators in the old and new problem states. He then wishes to use MEA to transform a previous solution of a problem similar to the current one into a complete solution of the current problem.

As a difference function in this transformed MEA problem, he proposes using the _same_ difference function as is already used to compare the initial and goal states in a conventional MEA approach to the current problem; this difference function now becomes a "similarity metric" between different problems[30]. Having found an analogous problem, i.e. one with a high degree of similarity to the current one, MEA is applied to reducing the difference between this problem and the current one, thus leading to a solution of the new problem derived by analogy with the old. Thus MEA is being applied not to the current problem and its goal, but to the current problem/goal and a previ-

---

29 A _path constraint_ is a rule which prohibits certain operator sequences even though they made produce a solution, e.g. because the solution thus arrived at may be too costly.

30 It is not, in fact, necessarily the case that this function be a "metric" in the strict mathematical sense; "measure" would be a more precise term.

ously solved problem/operator-sequence/goal.

A number of (meta-)operators are proposed as useful for this higher-level MEA problem; these include insertion and deletion of operators from a sequence, adding new operator sequences at the start or end of a sequence, reordering operators, and "meshing" of two operator sequences - the last of these is considered as being in itself "an interesting and potentially complex problem" .

The difference function between states of the transformed problem is a 4-tuple comprising the differences between problem states, goal states, path constraints and operator applicability. In general, it will not always be possible to reduce one component of this 4-tuple without at the same time increasing another. One possible way of avoiding this difficulty is to try always to reduce some linear combination of the four components.

In order to make possible the retrieval of problems similar to the current one, it is clear that some form of memory organisation based upon similarity of problem states is required. The solving of a problem by analogy naturally leads to the assimilation of the new problem within the existing structure; thus the activities of problem-solving, learning and analogisation are deeply linked. The structure of an "episodic" memory such as is required is regarded by Carbonell as "relatively simple";

we would regard this as by no means self-evident, and would consider the development of a large practical program embodying Carbonell's ideas in a domain with a large collection of previously solved problems and of possible operators as a major achievement. There is an obvious danger that the search for an analogous problem will prove to be non-trivial, so that once again one has merely substituted one form of search for another. A possible starting point for a large-scale implementation of Carbonell's ideas may be work such as that of Cohen [Cohen1980a] on an intelligent theorem prover which attempts to use theorems already proven as a guide to the proof or refutation of new conjectures.

The Use of Similarity Measures in Retrieval and Assimilation

We have seen in the last few pages reference to the use of similarity measures in the evaluation of the strength of analogies. However, there has been no suggestion that such measures might actually be used as a basis for knowledge organisation and retrieval. We describe here, in rather abstract terms, a possible use of similarity measures on formal structures for the large-scale organisation of knowledge. The assumptions are (i) that the knowledge to be organised can be divided (perhaps quite arbitrarily) into structured units (e.g. "concepts") and (ii) that there exists a collection of partial metrics[31] $\{\partial_i\}$ upon these units which measure the degree of dissimilarity between them in various respects. There are no assumptions about the type of structure used (which could, for example, be a labelled directed graph representing a semantic net, or a collection of predicate calculus clauses), nor about what specific features are to be used to determine similarity; it is, however, highly

---

31 A *partial metric* is a function ∂ such that:

$$\forall x \; \partial(x,x)=0$$
$$\forall x \forall y \; \partial(x,y)=\partial(y,x)$$
$$\forall x \forall y \forall z \; \partial(x,y)+\partial(y,z) \geq \partial(x,z)$$

We do not require the condition

$$\forall x \forall y \; \partial(x,y)=0 \Rightarrow x=y$$

desirable in practice that the measures be cheap to compute.

These measures can be seen as defining a "feature space", in which the distance between two points is a metric derived from the set of similarity measures (e.g. a Cartesian metric: the square-root of the sum of squares of the similarity measures). Carbonell [Carbonell1981a], for example, uses a set of four differences derived from a Means-Ends Analysis of a problem to define a distance between two problems, as discussed above. The use of a feature space has some affinity with the technique known as multidimensional scaling, in statistical taxonomy ([Green1972a]). There are, however, two significant differences. The first is that in multidimensional scaling, the goal is to reduce a large set of coordinates (i.e. a many-dimensional space) to a smaller set of linear combinations of these (i.e. a space of fewer dimensions), onto which a _previously given_ set of data may be projected with minimum loss of information. That is, the object of multidimensional scaling is to induce from given data what set of features may best be used to classify them. In contrast, we are supposing that the _classification_ be given (the similarity measures), and that the data (which are potentially any items of representable knowledge whatsoever) be _not_ all explicitly available at the outset. The second difference is that the features normally considered in multidimensional scaling are <u>scalar</u>; that is, each corresponds to a single numerical coordinate. In our model, this need not be the case at all; there is no reason to assume that, for some measure ∂ and objects A, B and C, that $\partial(A,B) \simeq \partial(A,C)$ implies that B is close to C.

Our task, then, is to find a way of locating the close neighbours of some new datum amongst an existing knowledge base. For simplicity, we shall consider only a single similarity measure, ∂.

We suppose that there is some set σ of points, and some distance δ_1, such that for all already known points X, there is a point Y in σ such that $\partial(X,Y) < \delta_1$. It is clear that such a set can be chosen; we consider the points of σ as <u>representatives</u> of regions of the feature space. Formally:

Let $K = \{\text{known points}\}$

$$\forall X \varepsilon K \; \exists Y \varepsilon \sigma \quad (\partial(X,Y) < \delta_1)$$

δ_1 is chosen sufficiently large that σ is small. Suppose we wish to assimilate a new point, Z. Then the first step of the algorithm is to measure $\partial(Z,Y)$ for each Y in σ. If each such measure is greater than δ_1, then it is clear that Z is further than δ_1 from every known point; in this case we add Z to σ, and conclude that it has no close neighbours. Otherwise, Z belongs in the neighbourhood of some representative, say Y. This algorithm is applied recursively; this requires that with each point in σ there is associated a set of representatives covering its neighbourhood to within a distance $\delta_2 (<\delta_1)$, and so on. Hence, we use the computed distances to organise the feature space into a hierarchical set of neighbourhoods; then a new datum is assimilated by a process of "homing in" on ever smaller neighbourhoods until either we find other, sufficiently similar data, or discover that there are none.

Formally, again,

Let $\delta_1 > \delta_2 > \ldots > \delta_n$

Choose a hierarchy of sets:

$$\sigma \text{ s.t. } \forall X \varepsilon K \ \exists Y \varepsilon \sigma \ \partial(X,Y) < \delta_1$$

$$\{\sigma_i : i \varepsilon \sigma\} \text{ s.t. } \forall X \ \partial(X,i) < \delta_1 \Rightarrow \exists Y \varepsilon \sigma_i \ \partial(X,Y) < \delta_2$$

$$\{\sigma_{i,j} : j \varepsilon \sigma_i\} \text{ s.t. } \forall X \ \partial(X,j) < \delta_2 \Rightarrow \exists Y \varepsilon \sigma_{i,j} \ \partial(X,Y) < \delta_3$$

etc.

Further,

$$\forall X,Y \varepsilon \sigma_{i_1,\ldots,i_j} \ \partial(X,Y) \geq \delta_j \qquad (*)$$

Given ξ

To find $S = \{\eta : \partial(\xi,\eta) < \delta_n\}$

Find $S_1 = \{i \varepsilon \sigma : \partial(\xi, i) < \delta_1\}$

If $S_1 = \emptyset$

then insert ξ into σ

return $\emptyset$

else Find $S_2 = \{S_i\}$

where $S_i = \{j \varepsilon \sigma_i : \partial(\xi, j) < \delta_2\}$

etc.

The sparseness given by the condition (*) ensures that this will lead to minimum search. In the early stages of knowledge acquisition, it will often happen that a new datum has no close neighbours; in this case we insert it at an appropriate level of the hierarchy. If any of the sets of representatives becomes too large, it can itself be split into a hierarchy.

One extreme of this approach is clearly to take δ_1 as zero; in this case the "hierarchy" becomes flat, and the algorithm is simply "compare ξ against every point in K". The other extreme is always to maintain the hierarchy as a binary tree. The first of these gives very large searches, but never requires a potentially expensive rebuilding of some part of the hierarchy; the latter leads to minimal searches but at the expense of frequently needing to add new points at high levels of the hierarchy.

We suggest that some algorithm based upon the above, in conjunction with a suitable collection of similarity measures such as those of Carbonell, could form a reasonable basis for the large-scale organisation of a knowledge base.

Summary and Conclusions

In the foregoing pages we have covered a wide range of material from the literature of AI, linked together by the common strands of relevance to mathematical discovery and analogical reasoning. It is, we believe, clear that not only have none of the works discussed "solved" the key problems in these areas, but that few of them have even achieved convincing solutions to those subproblems which they chiefly addressed. Whilst it is true that it is always easier to criticise destructively than constructively, to find defects than to point to positive achievements, it is nevertheless notable in how many of the works discussed there have been serious shortcomings.

This may sound like bleak pessimism, a counsel of despair. For if the combined intellects of dozens of distinguished workers in a field of enquiry cannot produce better solutions than this to a problem, must not the problem be close to insoluble? Our answer to this rhetorical question, however, is that such is not the case. It is indeed true that the problem of formulating a model of reasoning in which analogy plays a major role is extremely hard, whether psychological validity be sought or not. But progress has undoubtedly been made in a very diverse collection of relevant topics; we would point to the work of Munyer [Munyer1977b], Lenat [Lenat1976a], and R.Brown [Brown1977a] as being recent work of considerable value.

Lenat in particular, despite being open to criticism on a number of serious issues, has at least demonstrated that it is possible to build a program which is able to carry out a range of tasks in the exploration of a simple mathematical domain, including concept-formation and the proposal of hypotheses. He has abstracted a number of useful rules describing such a search process, and there is surely progress to be made from the incorporation of a similar body of "heuristics" within a cleaner framework, where the issue of flow of control and the details of implementation obtrude less upon the mechanism of the program.

Overall, we can distinguish two principal lines of attack on the problems of mathematical reasoning; loosely speaking, we may categorise these as "theorem proving" (exemplified by Munyer, R.Brown, Kling [Kling1971a], and Cohen [Cohen1980a]), and "rule-based system" (Lenat, Moore and Newell [Moore1973a], Langley). R.Brown has also looked at the problems of search-control in analogy matching in a way which naturally leads to consideration of the topic of "node labelling" on graphs, area well-known in other areas of AI (Waltz [Waltz1975a], Shneier [Shneier1978a], etc.). No doubt a truly intelligent reasoning program, if one is ever written, will make use of a mixture of all of these, together with others as yet unexplored.

A promising area for enquiry is that of using a rule-based system for controlling search; something along these lines forms part of Bundy's PRESS system [Bundy1981b] for symbolic algebra. Similar ideas are embedded within Lenat's AM, where some of the "heuristics" are in fact search control mechanisms, and Davis [Davis1979a] has proposed building an expert system to advise on search strategies within large problem spaces.

However, it must be re-iterated that the problems remaining are formidable. Indeed, as with many philosophical enquiries (and there is no doubt that much research in AI is at least as much a philosophical undertaking as it is an experimental and mathematical one), the outstanding difficulty remains that of formulating the questions.

References

Bledsoe1977a. W. Bledsoe, "Non-Resolution Theorem Proving", Artificial Intelligence Vol. 9(1) pp. 1-35 (1977).

Bobrow1977a. D. Bobrow and et al., "GUS: a Frame-Driven Dialog System", Artificial Intelligence Vol. 8(1)(1977).

Boden1977a. M.E. Boden, Artificial Intelligence and Natural Man, Harvester Press, Hassocks (1977).

Bono1967a. E. de Bono, The Uses of Lateral Thinking, Jonathan Cape (1967).

Boyer1979a. R.S. Boyer and J S. Moore, A Theorem-prover for Recursive Functions, SRI International (1979).

Bradshaw1980a. G.L. Bradshaw, P. Langley, and H.A. Simon, BACON.4: The Discovery of Intrinsic Properties, Proc. 3rd National Conf. of the Canadian Society for Computational Studies of Intelligence (1980).

Brown1977b. F.M. Brown, "Towards the Automation of Set Theory and its Logic", DAI Research Report no. 34, Edinburgh University (1977).

Brown1976a. R. Brown, "Reasoning by Analogy: A Progress Report", A.I. Working Paper 132, MIT (1976).

Brown1977a. R. Brown, Use of Analogy to Achieve New Expertise, MIT (1977). (M.Sc. Thesis)

Bundy1981a. A. Bundy, The Winston-Plotkin-Young-Linz Learning Program, Edinburgh DAI Prolog Program Library (1981).

Bundy1981b. A. Bundy and L.S. Sterling, Meta-level Inference in Algebra, Department of Artificial Intelligence, Edinburgh (1981). Working Paper 164

Bundy1982a. A. Bundy and B. Silver, A Critical Survey of Rule Learning Programs, Department of Artificial Intelligence, University of Edinburgh (1982). Research Paper no. 169

Bundy1982b. A. Bundy, Artificial Mathematicians: the Computer Modelling of Mathematical Reasoning, Academic (1982).

Carbonell1981a. J.G. Carbonell, A Computational Model of Analogical Problem Solving, Proceedings of IJCAI-7, Vancouver (1981).

Clocksin1981a. W.F. Clocksin and C. Mellish, Programming in Prolog, Springer (1981).

Cohen1980a. D. Cohen, Knowledge Based Theorem Proving and Learning, CMU Dept. of Computer Science (1980). (Ph.D. thesis)

Davis1979a. R. Davis, Seminar at University of Edinburgh, Dept. of Artificial Intelligence 1979.

Doran1966a. J. Doran and D. Michie, Experiments with the Graph Traverser Program, Proc. Royal Society (A) (1966).

Edwards1977a. H.M. Edwards, Fermat's Last Theorem: A Genetic Introduction to Algebraic Number Theory, Springer, New York (1977). Graduate Texts in Mathematics no. 50

Evans1967a. T.G. Evans, "A Heuristic Program to Solve Geometry Analogy Problems", in Semantic Information Processing, ed. M. Minsky, MIT (1967).

Farreny1982a. H. Farreny and H. Prade, About Flexible Matching and its Use in Analogical Reasoning, ECAI-82, Orsay (1982).

Fikes1972a. R.E. Fikes, P.E. Hart, and N.J. Nilsson, "Learning and Executing Generalized Robot Plans", Artificial Intelligence Vol. 3 pp. 251-288 (1972).

Freuder1978a. E.C. Freuder, "Synthesizing Constraint Expressions", CACM Vol. 21(11) pp. 958-966 (1978).

Green1969a. C. Green, "Theorem Proving by Resolution as a Basis for Question Answering Systems", in Machine Intelligence 4, ed. D. Michie & B. Meltzer, Edinburgh University Press (1969).

Green1972a. P.E. Green and V.R. Rao, Applied Multidimensional Scaling: A Comparison of Approaches and Algorithms, Holt Rinehart, New York (1972).

Hadamard1945a. J. Hadamard, The Psychology of Invention in the Mathematical Field, Princeton University Press (1945). (Reprinted by Dover, 1954)

Hanna1981a. F.K. Hanna and G.D. Ritchie, _AM: A Case Study in A.I. Methodology_, University of Kent Electronics Laboratories (1981).

Hedrick1976a. C.L. Hedrick, "Learning Production Systems from Examples", _Artificial Intelligence_ Vol. 7 pp. 21-49 (1976).

Hewitt1969a. C. Hewitt, _PLANNER: A Language for Proving Theorems in Robots_, Proceedings of IJCAI-1, Washington (1969).

Kling1971a. R.E. Kling, "A Paradigm for Reasoning by Analogy", _Artificial Intelligence_ Vol. 2 pp. 147-178 (1971).

Knapman1978a. J. Knapman, _A Critical Review of Winston's "Learning Structural Descriptions from Examples"_, AISB Quarterly no. 31 (1978).

Koestler1964a. A. Koestler, _The Act of Creation_, Hutchinson & Co. (1964).

Lakatos1976a. I. Lakatos, _Proofs and Refutations_, Cambridge University Press (1976).

Langley1978a. P. Langley, "BACON.1: A General Discovery System", CIP Working Paper No. 383, Carnegie-Mellon University (1978).

Langley1979a. P. Langley, _Rediscovering Physics with BACON.3_, Proceedings of IJCAI-6, Tokyo (1979).

Lenat1976a. D.B. Lenat, _AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search_, Stanford University (1976). (Ph.D. Thesis)

Lenat1977a. D.B. Lenat, _The Ubiquity of Discovery_, Proceedings of IJCAI-5, Boston (1977).

Lenat1981a. D.B. Lenat, ARPAnet communication to P.Hayes 1981.

Mackworth1977a. A.K. Mackworth, "Consistency in Networks of Relations", _Artificial Intelligence_ Vol. 8(1) pp. 99-118 (1977).

Michener1978a. E.R. Michener, _Representing Mathematical Knowledge_, MIT (1978).

Minsky1975a. M. Minsky, "A Framework for Representing Knowledge", in The Psychology of Computer Vision, ed. P.H. Winston,McGraw-Hill (1975).

Mitchell1981a. T. Mitchell, J.G. Carbonell, and R. Michalski, Special Issue on Machine Learning, ACM (SIGART) (1981).

Mitchell1978a. T.M. Mitchell, Version Spaces: An Approach to Concept Learning, Stanford University (1978). (Ph.D. Thesis)

Mitchell1979a. T.M. Mitchell, An Analysis of Generalization as a Search Problem, Proceedings of IJCAI-6, Tokyo (1979).

Moore1973a. J. Moore and A. Newell, "How Can Merlin Understand?", in Knowledge and Cognition, ed. L. Gregg,Lawrence Erlbaum Associates (1973).

Munyer1977a. J.C. Munyer, Towards the Use of Analogy in Deductive Tasks, Proceedings of IJCAI-5, Boston (1977).

Munyer1977b. J.C. Munyer, Analogy as a Heuristic for Mechanical Theorem Proving, MIT (1977). (Workshop on Automatic Deduction - extended abstract)

Newell1972a. A. Newell and H.A. Simon, Human Problem Solving, Prentice Hall, New Jersey (1972).

Nilsson1971a. N.J. Nilsson, Problem-Solving Methods in Artificial Intelligence, McGraw-Hill (1971).

Pask1975a. G. Pask, D. Kallikourdis, and B.C.E. Scott, "The Representation of Knowables", International Journal of Man-Machine Studies Vol. 17 pp. 15-134 (1975).

Piaget1954a. J. Piaget, The Construction of Reality in the Child, Basic Books, New York (1954).

PlatoBC360a. Plato, The Republic, Sphere (BC360). (translated by B.Jowett)

Plotkin1977a. G.D. Plotkin, Lecture in the Dept. of Artificial Intelligence, University of Edinburgh 1977.

Poincare1913a. H. Poincare, "Mathematical Creation", in The Foundations of Science, The Science Press (1913). (translated by G.B. Halstead)

Polya1945a. G. Polya, How to Solve It, Princeton (1945).

Polya1954a. G. Polya, Mathematics and Plausible Reasoning, Princeton (1954).

Polya1962a. G. Polya, Mathematical Discovery, Wiley (1962).

Polya1965a. G. Polya, Mathematical Discovery, Wiley (1965).

Popper1959a. K.R. Popper, The Logic of Scientific Discovery, Hutchinson, London (1959).

Popper1963a. K.R. Popper, Conjectures and Refutations, Routledge and Kegan Paul, London (1963).

Potschke1982a. D. Potschke, Toward a Mathematical Theory of Analogical Reasoning, ECAI-82, Orsay (1982).

Quillian1968a. Quillian, M.R., "Semantic memory", in Semantic Information Processing, ed. M. Minsky, MIT (1968).

Shapiro1982a. E. Shapiro, Inductive Inference of First Order Theories from Facts, Yale Computer Science Department (1982). (to appear as report no. 192)

Shneier1978a. M.O. Shneier, Object Representation and Recognition in Machine Vision, Edinburgh University (1978). (Ph.D. thesis)

Sussman1973a. G.J. Sussman, "A Computational Model of Skill Acquisition", TR 297, MIT (1973).

Vere1977a. S.A. Vere, Induction of Relational Productions in the Presence of Background Information, Proceedings of IJCAI-5, Boston (1977).

Waerden1971a. B.L. Van der Waerden, "How the Proof of Baudet's Conjecture was Found", pp. 251-260 in Studies in Pure Mathematics (Presented to Richard Rado), Academic Press, London (1971).

Waltz1975a. D. Waltz, "Understanding Line Drawings of Scenes with Shadows", in The Psychology of Computer Vision, ed. P.H. Winston, McGraw-Hill (1975).

Winston1975a. P.H. Winston, "Learning Structural Descriptions from Examples", in The Psychology of Computer Vision, ed. P.H. Winston, McGraw-Hill (1975).

Wittgenstein1953a. L. Wittgenstein, Philosophical Investigations, Blackwell, Oxford (1953).

Young1977a. R.M. Young, G.D. Plotkin, and R.F. Linz, Analysis of an extended concept-learning task, Proceedings of IJCAI-5, Boston (1977).

Zadeh1979a. L.A. Zadeh, "A Theory of Approximate Reasoning", pp. 149-154 in Machine Intelligence, Vol. 9, ed. L.I. Mikulich,Ellis Horwood (1979).