AM: A Case Study in AI Methodology

G.D. Ritchie*
Department of Computer Science, Heriot-Watt University, Edinburgh EH1 2HJ, U.K.

F.K. Hanna
Electronics Laboratories, University of Kent, Canterbury CT2 7NT, U.K.

Recommended by Pat Hayes

ABSTRACT
Much artificial intelligence research is based on the construction of large impressive-looking programs, the theoretical content of which may not always be clearly stated. This is unproductive from the point of view of building a stable base for further research. We illustrate this problem by referring to Lenat's AM program, in which the techniques employed are somewhat obscure in spite of the impressive performance.

1. Introduction

Artificial intelligence is still a relatively young science, in which there are still various influences from different parent disciplines (psychology, philosophy, computer science, etc.). One symptom of this situation is the lack of any clearly defined way of carrying out research in the field (see [11] for some pertinent comments on this topic). There used to be a tendency for workers (particularly Ph.D. students) to indulge in what McCarthy has called the "look-ma-no-hands" approach [4], in which the worker writes a large, complex program, produces one or two impressive printouts and then writes papers stating that he has done this. The deficiency of this style of "research" is that it is theoretically sterile—it does not develop principles and does not clarify or define the real research problems. What has happened over recent years is that some attempt is now made to outline the principles which a program is supposed to implement. That is, the worker still constructs a complex program with impressive behaviour, but he also provides a statement of how it achieves this performance. Unfortunately, in some cases, the written "theory" may not

*Present address: Department of Artificial Intelligence, University of Edinburgh, Edinburgh EH8 9NW, U.K.

0004-3702/84/$3.00 © 1984, Elsevier Science Publishers B.V. (North-Holland)

correspond to the program in detail, but the writer avoids emphasising (or sometimes even conceals) this discrepancy, resulting in methodological confusion. The "theory" is supposedly justified, or given empirical credibility, by the presence of the program (although the program may have been designed in a totally different way); hence the theory is not subjected to other forms of argument or examination. Anyone trying to repeat or develop the reported work may find that the written accounts are inconsistent or apparently incorrect, despite the fact that they have been presented as tried and tested. In this situation, the unfortunate person who wishes to continue in this field must return to the beginning, since the alleged "state-of-the-art" (i.e., his predecessor's "theory") gives very little guidance. It specifies a performance to be attained, but not how to attain it. What then has the original work achieved, in scientific terms?

We will illustrate this methodological phenomenon using an example of recent research in artificial intelligence. Lenat [7] has described a program ("AM") which "discovers" concepts and conjectures in elementary mathematics, and he claims that the program progresses from pre-numerical knowledge to that of an undergraduate mathematics student. Since Lenat was invited to give the 1977 *Computers and Thought* lecture [9], we assume that this work (hailed in [12] as a major advance) was held in high esteem by the AI community. Although Lenat makes some interesting proposals which can be assessed on their own merits, the substance of his research is based on the AM program. Close inspection of the written accounts of AM [7, 8, 10] suggests that there are some worrying discrepancies between the theoretical claims and the implemented program, and it is this aspect which we wish to consider here. We accept that the AM program was capable of producing the published output. What we wish to argue is that the written accounts (particularly [8] and the early chapters of [7]) give a misleading view of how the program worked, to an extent which obscures the real contribution of the work, and which may confuse others in the field.

Sections 2–4 will contain an examination of the account of the AM program given in [7]. This will involve a large amount of technical detail, but this is necessary to substantiate our main argument. In Section 5 we will return to the issue of research methodology and make some observations based on the technical discussion in the body of the paper.

2. The Official Overview

The behaviour of AM is neatly summarised by Lenat:

> "AM began its investigations with scanty knowledge of a hundred elementary concepts of finite set theory. Most of the obvious set-theoretic concepts and relationships were quickly found (e.g. de Morgan's laws; singletons), but no sophisticated set theory was ever

done (e.g. diagonalization). Rather, AM discovered natural numbers and went off exploring elementary number theory. Arithmetic operations were soon found (as analogs to set-theoretic operations) and AM made rapid progress to divisibility theory. Prime pairs, Diophantine equations, the unique factorization of numbers into primes, Goldbach's conjecture—these were some of the nice discoveries by AM." [8, p. 839]

The program initially contains data-structures representing about 100 elementary mathematical concepts (e.g. sets, composition of operations) and about 230 "heuristic rules". The data-structures are similar to the "frames" of Minsky [13] or Bobrow and Winograd [2], in that they are "concepts" containing a list of "facets" (slots) which can be filled with other data-items. The "heuristic rules" are like "productions" [14], in that each rule has a "test" (a conjoined list of truth-valued expressions) and an "action" (a list of executable statements, with side-effects). AM proceeds by trying to fill in facets of concepts, using the rules. The program is controlled by an "agenda" of "tasks", in order of priority. Each task-descriptor on the agenda includes a note of a particular "operation-type" (of which there are only two or three), a particular "facet-name" (of which there are about twenty) and a particular concept-name. That is, a task is always of the form

"Do operation A to facet F of concept C"

The task descriptor has other information (e.g. the amount of time and space allocated to it) but no associated program. (That is, it is not like an executable process description, but is more like a specification of a "goal" to be achieved.) The scheduler in AM always takes the top (highest priority) task from the agenda, and attempts to carry it out by finding suitable heuristic rules. Every rule has, as the first conjunct of its list of tests, a check to see what the operation-type, facet and concept of the current task are, and so each rule can be stored on exactly the facet to which it is appropriate. That is, every facet of every concept has a few "sub-facets" (one for each of the operation-types), containing lists of rules appropriate to tasks involving that combination of concept, facet and operation-type.

Having found the relevant rules by this indexing mechanism, the AM system applies each one by checking its "tests" and, if all these yield "true", executing its actions. This goes on until the collected group of rules are all applied, or the time/space allocation for the current task is exhausted.

The concepts are organised into a hierarchy of "specialisations" and "generalisations", with each concept having a facet to hold a list of known examples. In this way, heuristic rules can be stored only once for a general concept, and more specialised concepts (lower down the hierarchy) can inherit the general rules during the process of retrieving relevant heuristics. In general, any facet of a concept will be "inherited" by that concept's specialisations.

Each task has a priority rating, and every facet and concept in the initial data-structure has a "worth", representing how "interesting" it is. These numerical measures, and computations on them, are used to direct the flow of the program in useful directions. Newly created concepts are given "worths" according to various formulae in the rules which create them.

The program prints out various messages to inform the human user of its progress, and the user can interrupt at any point to request further information, or to give AM a limited amount of direction.

Starting with the initial base of about 100 concepts and 230 rules, AM "discovers" (i.e. constructs and categorises as "interesting") various mathematical concepts, including natural numbers, and several conjectures, such as the fact that numbers can be uniquely factorised into primes.

3. Assessing AM

It is extremely hard to evaluate work of this nature. Lenat [7] discusses AM from various standpoints, but most of his comments are based on the assumption that the program had "discovered" various mathematical concepts using a very simple, uniform, concept-based search procedure. He goes on to discuss further issues, such as the need for "meta-heuristics", (rules about rules) or how AM's performance compares with a human's (about "undergraduate math major", he estimates). We would like to discuss the more fundamental question of what it was that AM actually did. To be more specific:

(3.1) Was the control structure simply the uniform regime described in [7, Chapters 1–6] and in [8]? This is not a mere implementation detail, since the organisation of tasks, concepts and rules (outlined in Section 2 above) is Lenat's answer to the crucial question: how does the program choose what to do at any given point in its inferences?

(3.2) Were there procedures and statements in the program (other than the heuristic rules) which embodied important "knowledge" for the operation of the system?

(3.3) What did AM actually discover, and to what extent should it be assessed on its results (as opposed to its internal workings)?

We shall attempt to answer some of these questions in the following sections, using quotations from [7] wherever appropriate, but without listing all the evidence exhaustively. (All page numbers will be from [7] unless otherwise stated).

4. How Does AM Work?

4.1. Control structure

The description in [8] and in [7, Chapters 1–6] states, quite definitely, that AM has only the simple control structure outlined in Section 2 above. The scheduler takes the top priority task, finds the relevant rules via the operation/concept/facet triple, and executes these rules:

> "AM repeatedly selects the top task from the agenda and tries to carry it out. This is the whole control structure!" (p. 15).
>
> "The flow of control is simple: AM picks the task with the highest priority value, and tries to execute it." (p. 34).
>
> "What precisely does AM do, in order to execute the task? . . . The answer can be compactly stated as follows: 'AM selects relevant heuristics, and executes them'" (p. 35).

Also, the heuristic rules are claimed to be stored on the relevant Concept structures:

> "The secret is that each heuristic rule is stored somewhere *a propos* to its 'domain of applicability'. This 'proper place' is determined by the first conjunct in the left-hand side of the rule." (p. 54).
>
> "The key observation is that a heuristic typically applies to *all examples of a particular concept C* . . . This is in fact where the heuristic rule *is* stored" (p. 55).
>
> "Initially, the author identified the proper concept *C* and facet *F* for each heuristic *H* which AM possessed, and tacked *H* onto *C.F.* This was all preparation, completed long before AM started up. Each heuristic was tacked on to the facet which uniquely indicates its domain of applicability" (p. 55).

The relevant concept/facet pair for storing each rule is determined thus:

> "The first conjunct of the IF-part of each heuristic indicates where it is stored and where it is applicable" (p. 55).
>
> "In fact, AM physically attaches each rule to the facet and concept mentioned in its first conjunct" (p. 37).

The rules' first conjuncts should specify not only a concept/facet pair, but also an action:

> "The first conjunct will always be written out as follows, in this document (where *A*, *F*, and *C* are specified explicitly): The current task (the one just selected from the agenda) is of the form 'Do action *A* to the *F* facet of concept *C*'" (p. 37).
>
> ". . . each facet of each concept can have two additional 'subfacets' . . . named Fillin and Check. 'Fillin' field of facet *F* of

> concept *C* is abbreviated *C.F.*Fillin. The format of that subfield is a list of heuristic rules, encoded into LISP functions. Semantically, each rule in *C.F.*Fillin should be relevant to filling in entries for facet *F* of any concept which is a *C*" (p. 101).

It must be emphasised that these quotations are not taken misleadingly out of context; the written accounts (e.g. [8]) do give a very definite impression that the principle claim being advanced for the AM program was that a simple, uniform control structure does in fact produce the impressive output already alluded to. Closer inspection of the thesis and (more particularly) the rules in the Appendix 3 reveals that the AM program did not actually function in this way.

There is some confusion about the "subfacets" on which rules may be entered. Sometimes (e.g. p. 101), two subfacets (Check and Fillin) are listed, whereas other remarks (e.g. p. 67) refer to three (Check, Fillin and Suggest). On p. 41, there is the remark (pertaining to task-types):

> "Each possible Act *A* (e.g. Fillin, Check, Apply, etc.)..."

which seems to imply a wider set than just Fillin and Check. ("Apply" is not referred to elsewhere.) The rules in the Appendix are categorised under four subheadings—Check, Fillin, Suggest and Interest. Of these, only Check and Fillin fit clearly into the generalisation of being actions that define a task (as in the quotations from p. 37 and p. 101 given above). The "Suggest" rules are described (p. 100) as rules to be used when there are no sufficiently interesting tasks on the agenda, but no explanation is given as to why they should, in this case, be attached to concepts or facets. Obviously, it would allow the Suggest rules to be scanned by running through all the concepts in the system, but this could be achieved by having a simple global list of all Suggest heuristics in the system. No details are given of how this search occurs (e.g. from the top of the Concept hierarchy), despite the fact that this is a fairly central question when considering AM's decision process. ("AM may call on all the Suggest facets in the system" (p. 100).) Also, it is still not clear whether "Suggest" is a facet (containing the rules associated with a concept) or a subfacet (as implied on p. 226).

The "Interest" rules do not seem to fit into the scheme of being a rule "to do *A* to facet *F* of *C*", since they state aspects of a structure which make it "interesting"; e.g.

> "An operation is interesting if its values are interesting."

On p. 67, a facet is defined thus (for any concept *C*):

> "Interestingness: what special features make *C* especially interesting?"

The casual reader might think that this had something to do with "Interest" rules, since all the "Interest" rules given (in [7]) are under headings like "Operation.Interest", "Composition.Interest" ("*C.F*" being the notation for facet *F* of concept *C*). However, pp. 97–99 describe the contents of the "Interest" facet, and these are "Interest features" (not heuristic rules). It is left to the reader to make the following deduction:

- Interest rules compute Interest values for a Concept.
- There is (p. 67), a facet "Worth" which rates how interesting a Concept is.
- Hence, Interest rules can be looked on as filling in (or altering) the "Worth" of Concepts.
- Hence, these rules should be stored in the "Fillin" or "Check" subfacets of the "Worth" facet.

(I.e. the many headings in the appendix of [7] of the form "*C*.Interest" are a shorthand for "*C*.Worth.Fillin".)

Many rules in the appendix are listed against a subfacet of a *concept*, not of a facet of a concept. (For example, "Insertion.Check", "Coalesce.Fillin", "Canonise.Suggest".) This seems to suggest that a concept may have a group of Check, Fillin and Suggest heuristics which are not associated with any facet in particular. There is also the interpretation that such entries are really rules stored on the Examples facet for that concept; that is, subheadings such as Insertion.Check, Coalesce.Fillin are abbreviations for Insertion.Examples.Check, Coalesce.Examples.Fillin, respectively. As it happens, almost all the rules given in [7] are under subheadings which do not specify a facet-name. Only the rules associated with "Any-concept" have full subheadings of the form "*C.F.A*" (e.g. Any-concept.Analogies.Fillin). Hence all the Fillin and Check rules (apart from those on Any-Concept) are in fact (if we adopt the convention suggested here) attached to the Example facet of the Concept—other facets have no Check or Fillin rules. Further confusion is introduced on p. 230, where there is a comment about "Any-concept":

> "This concept has a huge number of heuristics. For that reason, I have partitioned off—both here and in AM—the heuristics which apply to each kind of facet. Thus the LISP program has a separate concept 'Examples-of-any-concept', another concept called 'Definitions-of-any-concept', etc."

This renders the concept of a "concept" even less clear.

So far, the arrangement is still fairly systematic. We have to modify the statements (in the quotations above) slightly so that:

(4.1.1) Interest heuristics are stored on the Fillin subfacets of the "Worth" facet. That is, they are invoked when a task is executed which is trying to Fillin the Worth of a Concept. (Some may also be on the Check subfacet of the Worth facet).

(4.1.2) Suggest heuristics are used when AM is trying to generate tasks for the agenda. They are stored on subfacets and on concepts, but the way of using this attachment, or of choosing which rules to invoke (i.e. the way of scanning the Concept hierarchy), is not specified.

(4.1.3) Heuristics described (in [7, Appendix]) as being on *C*.Check or *C*.Fillin are in fact stored on the corresponding subfacet of the Examples facet of *C*.

(In some parts of [7], (e.g. p. 178), Concepts are listed with facets called "Fillin", "Check", "Suggest" and "Interest", but this is presumably just an expository convenience, since it does not accord with usage elsewhere.) This still leaves one or two loose ends regarding the way that rules are stored and indexed. Some of the heuristics (Section 3.1 of the Appendix) are said to be for dealing with "any item *X*, be it concept, atom, event, etc. . .". It is not clear where these could be indexed, since the rule-retrieval routines described elsewhere operate only on the hierarchy of concepts, not "atoms" or "events". (The idea of an "event" is not mentioned anywhere else.) These rules appear to be generalisations that have been included in the written description in order to summarise the effect of *various* sections of the LISP program, rather than statements of actual individual rules which AM contained in the form stated in the thesis.

The whole notion of a "Concept" is confusing. In the descriptions of how the AM program operates (e.g. pp. 14–15, pp. 28–29, pp. 42–45, most of Chapter 5), a "Concept" seems to be defined as a particular kind of data structure, having about 20 "facets" (Generalisations, Specialisations, Examples, Algorithms, etc.) and representing a mathematical idea. Most of the written accounts seem to be based round this notion. However, on pp. 105–106 of [7], there is a diagram of "AM's Starting Concepts" which depicts a hierarchy including, at the top level, a universal class called "Anything" which is divided into "Any-concept" and "Non-Concepts". The question then arises—what are "Non-Concepts", in this sense? Do they have "facets"? If not, how do they fit into the hierarchy? If they do have "facets", in what sense are they not "Concepts" as defined in the earlier chapters? Do they have rules attached to them? If so, how? Are there two definitions of "Concept" in use? If so, what statements apply to which?

There are further confusing aspects concerning the invocation of rules. Many of the written rules do not have first conjuncts of the form "if current task is 'do *A* to facet *F* of *C*'", but have conditions that seem incompatible with the simple control regime given in the main part of [7]. For example, many rules are of the form "After . . .", and some refer to rather ill-defined aspects of the computation:

> "After verifying a conjecture for concept *C*. . ." (Rule 71).
>
> "If operation *F* has just been applied and has yielded a new concept *C* as a result . . ." (Rule 154).

> "If the structure is to be accessed sequentially until some condition is met, and if the precise ordering is superfluous..." (Rule 236, listed under "ordered-struc.check").
>
> "If the current task involves a specific analogy, and the request is to find more conjectures..." (Rule 78, listed under "any-concept.analogies.fillin").

An obvious interpretation is that Lenat has adopted [7, Appendix] an abbreviatory convention whereby the first conjunct of a rule (i.e. the part which would be of the form "Do A to F of C") is omitted, since the sub-heading within the list of rules contains that information (e.g. "Any-concept.Defn.Check"). Section 3.4 of the Appendix notes such a convention explicitly for the rules in that section, but it is not adhered to throughout the appendix, since some of the rules include a first clause which echoes the subheading. However, this flexibility probably helps to express the rules more fluently. Even if we assume that suitable first conjuncts are to be appended where necessary, the rules cited above (and some others) then have puzzling *second* conjuncts; for example, Rules 71 and 154 need some details of past history, Rule 236 describes a condition that is not, on the face of it, computable, and Rule 78 seems ambiguous between a task to "fill-in" analogies and one to "fill-in" conjectures.

Further evidence that the implemented control structure was not quite that described in the main chapters of [7], comes from some less noticeable areas of that document:

> "Unfortunately, AM's heuristics are not all coded as separate LISP entities, which one could then 'trace'. Rather, they are often interwoven with each other into large program pieces" (p. 140).
>
> "Note that although the Fillin and Suggest heuristics are blended into the relevant facets (e.g. into the Algorithms for Compose), the Interestingness type heuristics are kept separate, in this facet." (p. 220).

The latter quotation is in a footnote to a sample of AM program in which various "rules" appear as directly-called pieces of LISP (including Rule 154 cited above); that is, the "blending in" is not just the attachment of a procedure to a "subfacet", but involves actually putting the heuristic commands in the body of the facet's entry (the algorithm, in this example).

Many of the "rules" given in [7] seem to be justifications or summaries of computations that AM does as a result of various pieces of program within it, and which Lenat therefore regards as "knowledge" that AM has. That is, the "knowledge" is not built into actual rules, but is an interpretation which can be placed on various procedures and statements in the LISP implementation. For example:

> "If structure *S* is always to be maintained in alphanumeric order, then AM can actually maintain it as an unordered structure, if desired." (Rule 237).
>
> "The user may sometimes indicate 'Conjunction' when he really means 'Repeating'" (Rule 224).

The queries raised in this section are not minor organisational matters or implementation details. The whole flow of "reasoning" in AM is embodied in the execution of the rules, and how AM chooses what to do at any given moment depends entirely on the way that the rules are invoked. Otherwise, what claims or proposals does the thesis make about guiding heuristic search? If the implemented version is simply a large LISP program in which various procedures call each other in a partly pre-programmed way, it cannot be said that the control structure given in [8] (or in [7, Chapters 1–6]) has actually been tested on the subject matter described.

4.2. Use of other information

It would, of course, be impractical for Lenat to publish the full LISP code for the heuristics, and using English summaries of the rules is an obvious alternative. This has the possible weakness of introducing difficulties of interpretation for the reader, so it is important to make allowances for the imprecision of the medium (English) when assessing the published version. However, there are some instances where some work which is significant (and difficult to reconstruct by guess-work) is covered by some brief phrase. For example:

> "A nonconstructive existence conjecture is interesting" (Rule 74).
>
> "If two expressions (especially: two conjectures) are structurally similar, and appear to differ by a certain substitution, then if the substitution is permissible we have just arrived at the same expression in various ways, and tag it as such, but if the substitution is not seen to be tautologous, then a new analogy is born. Associate the constituent parts of both expressions. This is made interesting if there are several concepts involved which are assigned new analogues." (Rule 218).
>
> "Concept *C* is interesting if some normally-inefficient operation *F* can be efficiently performed on *C*'s" (Rule 22).
>
> "... replace the value obtained by some other (very similar) value..." (Rule 129).
>
> "If AM just referenced *X* and almost succeeded, but not quite, then

> look for a very similar entity *Y*, and retry activity with *Y* in place of *X*" (Rule 4, with the comment "There is a separate precise meaning for 'almost succeed', 'similar entity' and 'retry' for each kind of entity and activity that might be involved").

> "Formulate a parameterised conjecture, a 'template', which gets slowly specialised or instantiated into a definite conjecture" (Rule 69).

The problem with these obscurities is that they hide what are probably the most crucial parts of the "discovery" process. Given the discussion in Section 4.1 above, it seems that the use of a hierarchy of concepts together with heuristic rules indexed to these structures, is not the main mechanism of discovery—much non-trivial work occurs in the procedures called by these rules. Ways of computing "non-constructiveness" or "efficiency" or "similarity" are interesting and (presumably) make a significant contribution to the discovery process.

Sometimes the text admits that the procedures involved are "hacks" or "special-purpose":

> "... symbolically instantiate a definition... " (Rule 31, with the comment "simply says to use some known tricks, some hacks").

> "... it's worth creating more boundary examples and boundary non-examples by slowly transforming *x* and *y* into each other." (Rule 39, with comment "The rules for this slow transformation are again special purpose").

> "... ensure that every scrap of *C*.Defn has been used." (Rule 58, with comment "this rule contains several hacks (tricks) for checking that the definition has been stretched to the fullest").

> "Examine *C* for regularities" (Rule 67 in its entirety, with the comment that this contains "... a few special-purpose hacks and a few ultra-general hacks").

It is not clear what the reader is to infer from these deprecating comments. Are the rules over-simple, theoretically uninteresting, badly programmed, or unsatisfactory in some other way? Whatever the implication of dismissive epithets such as "hack" and "special purpose", the work which these procedures carry out seems to be fairly substantial, and without further explanation the reader has to accept an essentially subjective judgement (from Lenat) on these points.

Some of the information used by AM is far from obvious. Consider Rule 2:

> "If the user has recently referred to *X*, then boost the priority of any tasks involving *X*."

Hence, if the user re-names a concept, AM will devote more resources to examining it (as in Task 44 on p. 300). This means that re-naming (as shown in all the sample runs) is not a purely notational alteration, but represents advice from the user.

Many of the rules allude to data-structures, markings on structures, or scheduling procedures which are not fully explained and which seem to be additional to those described in the main part of the thesis (e.g. footnote 5, p. 239). Some of these are simple data-management rules (concerning the space occupied by a structure, and guidelines for disposing of neglected concepts, etc.), but others seem to contribute to the substance of the inference:

> "This is a slight extra boost given to each new operation, predicate, etc. This bonus decays rapidly with time..." (comment on Rule 144).
>
> "After dealing with concept *C*, slightly temporarily boost the priority value of each existing task which involves an Active concept whose domain or range is *C*." (Rule 14).
>
> "This rule is tagged as explosive, and is not used very often by AM." (comment on Rule 152).
>
> "The formula also incorporates a small factor which is based on the overall value of coalescings which AM has done so far in the run" (comment on Rule 202).
>
> "... the interestingness of *F* is weakly tied to that of *C*..." (Rule 173, with comment "If the new concept becomes very valuable, then *F* will rise slightly in interest. If *C* is so bad it gets forgotten, *F* will not be regarded so highly").

These last few obscurities tend to suggest that the actual mechanism used by the implemented program has not been fully explained (quite apart from the omissions and confusions detailed in Section 4 above). This is not to suggest that these mechanisms were pernicious or unsatisfactory, but that the amount of detail given is not appropriate to their apparent importance.

4.3. AM's discoveries

It would be possible to carry out AI research in which the value of the research could be assessed quite independently of the results produced by some particular program (see the discussion in Section 5 below). However, the impression given by the published accounts of AM is that the program produced significant discoveries, and much of the public impact of the system (particularly outside the AI research community) derived from the claims made (e.g. the quotation at the beginning of Section 2 above). For example, Lenat states that AM discovered Goldbach's conjecture (i.e., that every even number is the sum of two primes). Although one of the published runs (pp. 25–26) seems to illustrate this, Bundy [1, Chapter 13] observes that the example on p. 312 (which is also labelled as "Goldbach's conjecture") seems to illustrate a much more trivial conjecture (namely, that every even number is the sum of *some* primes, e.g., $2+2+\cdots+2$). It may well be that the designer of AM feels that no importance should be attached to what AM actually did, but the published accounts certainly place great emphasis on the program's performance. It is pertinent, therefore, to examine this performance in more detail.

One of AM's more striking discoveries was that of "natural numbers". As can be estimated from the graph on p. 123, and as Lenat comments (p. 144, footnote 22), most of the later "discoveries" depend on this one; if numbers are not found, AM cannot go on to further triumphs (e.g. the Unique Prime Factorisation conjecture). Let us look more closely at the steps involved in this particular discovery.

The following is an outline of how AM makes the discovery:

(4.3.1) The concept "object-equality", which is roughly the LISP notion of recursive "equal" (i.e. two lists are equal if all the elements are "equal", with atoms being equal if they are the same atom), can be generalised in two ways. If the recursive checking of each element (each CAR) is omitted, we get "equal-except-cars" (which AM calls "Genl-obj-equal"); if the recursive checking of the tail of the list (each CDR), we get "equally-nested-first-element" (which AM calls "Genl-obj-equal-1").

(4.3.2) There is a concept "Canonise" in the initial data for AM, which, when given two predicates P1 and P2, produces a function F mapping elements of the domain of P1 to elements of the domain of P2, with the condition that for all x, y in domain of P1, $\text{P1}(x, y)$ iff $\text{P2}(F(x), F(y))$. AM tries to form this function F for P1 = Genl-obj-equal-1, P2 = Obj-equal, and also for P1 = Genl-obj-equal, P2 = Obj-equal. (Notice that Genl-obj-equal is effectively a "same-length" test for two lists, so this latter attempt is trying to find an operation which will map lists of the same length to objects which Obj-equal will class as "equal".) The F constructed (to canonise Genl-obj-equal and Obj-equal) maps an object (list or bag) to a bag which contains a copy of the constant atom "T" for each element in the original structure.

(4.3.3) AM constructs an analogy between bag-operations (e.g. bag-union) and operations on these "canonical bags", and explores this (i.e. constructs the analogous operators). The canonical bags (of "T"s) can be regarded as natural numbers, and the operators constructed under the analogy correspond to addition, subtraction, etc.

Let us explore this sequence in more detail. The description given (p. 196) for initial concept "Canonise" is puzzling. One of its definitions is simply a statement (in English) of the definition given above (4.3.2), with no indication of how to execute it (e.g. how to verify the "iff" clause). The other two definitions use "Canonise.Algs" (i.e. the algorithms for Canonise), but Canonise has *no* algorithms—the description merely notes that heuristic rules will produce and fill in canonisations in various ways (that is, there is no recipe for making canonical forms—they are a consequence of the interaction of several rules).

How does Canonise come into the computation? This is extremely obscure. After AM creates the two generalisations of Obj-equal, and filled in some of their examples, there is a lull, since there are no sufficiently "interesting" tasks on the agenda (p. 329). AM therefore invokes "Suggest" heuristics to find something to do. As noted in Section 4.1 above, it is never explained what the selection procedure is for knowing what Suggest rules to use. After these deliberations, the agenda has two new tasks at positions 1 and 2; these are "Canonise Genl-obj-equal-1 and Obj-equal" and "Canonise Same-size and Obj-equal" (the user has just renamed Genl-obj-equal as "Same-size"). These have been suggested by Rule 213. AM goes on to construct a canonical connection as requested by the first task, using (presumably) Rules 206 and 207, and produces the set of lists which are either atoms or lists of one atom (apparently—the commentary does not explain this in detail). AM also posits an analogy between Objects and Canonical-Objects, but this does not seem to be followed up. In fact, this whole thread of exploration seems to be neglected, despite the fact that it starts as a wholly symmetrical partner alongside the examination of Gen-obj-equal (Same-size). Why is one followed and not the other, particularly as Gen-obj-equal-1 is actually further developed (e.g. canonised with respect to Obj-equal) first? There is a hint (p. 290) that this has something to do with the lack of variety of lists known to AM (with respect to their behaviour under these comparison operations) but this is not fully explained. (The user does re-name Gen-obj-equal as Same-size, which may boost its interest value, as commented in Section 4.2 above). The process of canonisation is repeated for the other pair (Same-size and Obj-Equal) again using Rules 206 and 207. These rules look distinctly ad hoc. Rule 206 contains a "special hand-crafted piece of knowledge" to tell AM how to canonise bags into bags-of-Ts, and Rule 207 is a "special-purpose rule . . . used to guide a series of experiments". The "worth" of the resulting canonisation function is given by Rule 208, for which "in the actual LISP code, an extra small term is added which takes into account the overall value of the Canonisations which AM has recently produced". Once again an analogy is posited between Bag-strucs and Canonical-bag-strucs, using Rule 209 (which says, roughly, that once a canonisation has been constructed, specific analogies should be tried). Analogies are one of the less satisfying parts of AM: "The Analogy facets are not implemented in full generality in the existing LISP version of AM." (p. 82). The exploration of this analogy leads, by various rules, to operations on Canonical-bag-strucs (the "natural numbers") which correspond to arithmetic manipulations.

None of the sample traces show exactly what rules are used at what points, so we can only speculate about the details of these runs. However, it is possible to gain the impression that the sucessful "discovery" was the result of various specially-designed pieces of information, aimed at achieving this effect. That fact alone would not vitiate AM's contribution to AI research. Suppose that someone were to design the relevant heuristics for this chain of discoveries by trying to devise a systematic route from sets, etc. (the initial concepts) to "canonical bags"; that is, by specifically attempting to make explicit a logical sequence of steps from a chosen origin to a definite destination. (There is a hint (p. 3) that this would be one way to design AM.) If this exercise were successful, then that would in itself be interesting, since we would have a detailed, machine-tested, mechanical inferential route from non-numeric concepts to the natural numbers. Such a result is not supposed to be the outcome of the AM project, and the written accounts do not propose this approach. For example, there are few details of rules such as 206, 207, 208 (see above), which would be part of the central body of the research under such a strategy, as the whole value of the work would be in these very details. The written accounts (e.g. [8]) do not seem to be putting forward such a result (i.e. a careful mechanisation of one or two chosen paths between concepts), and there is the statement [7, p. 18] that the knowledge base is not "finely tuned to elicit this one chain of behaviors". It is therefore probably not appropriate to try to assess AM in this light.

The question of "rating" of discoveries is also slightly under-explained, since Lenat's description simply classes the new concepts and conjectures into "winners" and "losers", without giving further details (e.g. the numerical interest ratings).

In a run starting with 115 concepts, AM developed 185 more concepts, which are classed (p. 125) as containing about 25 winners, 100 acceptable concepts, and about 60 losers. A full list of "losers" is not provided, and the list (pp. 224–225) of about 70 acceptable concepts does not show numerical ratings or distinguish the "winners". There are very few comments on the distinction between the two main classes (winners and losers). For example:

> "Thus AM did not develop the sixty 'losers' very much: they ended up with an average of only 1.5 tasks relevant to them ever having been chosen. The 'winners' averaged about twice as many tasks which helped to fill them out more. Also, the worth ratings of the losers were far below those of the winners." (p. 141).

> "Unfortunately, the sixty or seventy concepts which were losers were *real* losers" (p. 139).

Another pertinent question is: did AM ever form a concept which a human could regard as a "winner", and yet class it as uninteresting? That is, how many of the lower-rated results were actually valid mathematical concepts? If AM did underrate concepts or conjectures that are actually fairly interesting, this would not in itself be a great failure. However, if such misjudgements were combined with converse errors (i.e. overrating trivial items), then this would suggest that the system of interest-rating was not very accurate on the whole. This question is not fully discussed, although it is mentioned (p. 287) that "some concepts were created which were interesting to *us*... but which AM never bothered to develop".

This lack of information is not as crucial as that discussed in Sections 4.1 and 4.2 above, since the interest-ratings indicate AM's degree of "success", which is perhaps not as fundamental as the question of how AM actually worked.

5. Methodological Consequences

The AM project is a substantial piece of work in AI, and seems to contain a large amount of originality, creativity and detailed effort. We do not intend to detract from Lenat's interesting demonstration, but we believe that the published accounts of AM exemplify a style of scientific reporting within the AI community which is not conducive to continuity and the gradual development of a theoretical framework. The central point in our criticism is the apparent discrepancy between published claims (about performance and techniques) and implementation details. (The criticism is aimed entirely at the accounts of the AM program, and we are not discussing the question of whether Lenat could repair these deficiencies by writing another program which conforms more closely to his outline.) Our doubts can be summarised thus:

(5.1.1) Is the control structure simply the *uniform, minimal* mechanism given in [8], with the current task defining exactly what rules to execute?

(5.1.2) Are the data-structures *uniformly* represented as Concepts, with Facets and Subfacets as defined?

(5.1.3) Are there no "extra" processing mechanisms which have significant effects?

(5.1.4) How much of the real discovery work is buried in unexplained procedures?

(5.1.5) If the mathematical discoveries are really an important aspect of the work, can they be shown to have resulted *directly* from the search procedure as described?

In view of the pre-scientific nature of AI, it is always hard to assess the exact contribution of a piece of research, and it is necessary to be very careful in deciding what AM has contributed to progress in artificial intelligence. It would be unrealistic to demand that an AI project conform to the (rather idealised) notion of scientific investigation which is sometimes outlined in textbooks on the philosophy of science (e.g. [5, 15]). We are still at an early stage in the development of our scientific methodology, and very little of AI research fits into the traditional "experimental paradigm" in which well-defined hypotheses are refuted by empirical investigations. Let us weaken the demands for "scientific method", and try to formulate some realistic guidelines for AI research.

There are various ways in which the typical AI project (e.g. a doctoral thesis) could contribute usefully to our collective knowledge:

(5.2.1) It could introduce, in outline, a novel (or partly novel) idea or set of ideas. For example, ideas such as "consider a semantic structure as a computational procedure" (cf. [16]), or "regard computation as a sequence of messages between autonomous entities" (cf. [6]) are examples of general approaches which have been influential within AI. This kind of contribution could perhaps be forced into the mould of a scientific hypothesis by appending the predicate "... is a useful idea" on to each proposed concept, but this would not add much real rigour, since this variety of informal proposal is usually not precise enough to admit definite empirical refutation (nor is the notion of "useful" very precise). The refutation of the "usefulness" would come not from a specific critical experiment, but either from a general feeling that the concept in question had failed to form the basis of other "useful" work (despite various attempts), or from some theoretical argument as to its incoherence (as Fodor attempted for "procedural semantics" [3]). These may not be very striking or well-defined "refutations", but they are the best we can manage at present.

(5.2.2) It could elaborate the details of some approach. That is, starting from some idea (of the kind discussed in (5.2.1)), the research could criticise it, or fill in further details, in order to transform a slogan-like idea or metaphor into a fuller theory. This activity is comparable to theory-construction in a traditional science, but it is not directly tied to some standard means of testing, as will be discussed below.

(5.2.3) It could apply some articulated theory to some actual subject matter or data, and report the consequences. This is the nearest counterpart to traditional "experiment", but it differs in some important respects. Typically, the practical investigation proceeds by writing and running a computer program, and the assessment of the result of this activity is difficult, since AI ideas tend not to be formulated in such a way as to allow specific success/failure judgements.

An empirical AI "experiment" as in (5.2.3) can be scrutinised in various ways:

(5.3.1) Experiment design (static): does the structure of the program reflect the theory it purports to test?

(5.3.2) Experiment design (dynamic): does the internal processing behaviour of the program correspond to the dynamic aspects of the theory (if any)?

(5.3.3) Consistency: has the program been successfully run (repeatedly)?

(5.3.4) Results: what were the consequences, in terms of internal behaviour and in terms of output, of the program's execution?

(5.3.5) Interpretation: what do these results mean in terms of the theoretical constructs?

(5.3.6) Conclusions: what are the consequences for the theory of the interpreted results?

Much AI work contributes under all three categories (5.2.1)–(5.2.3), with most of the effort being expended on (5.2.2) and (5.2.3). We would accept that the AM project has contributed in respect (5.2.1); there is the proposal that mathematical discovery proceeds as a form of inference using domain-specific rules indexed to a hierarchical arrangement of concepts. The substance of the (5.2.2) contribution is less clear, since there is no separate statement of the theory apart from the description of the program, and Lenat [8, p. 834] dismisses many of the program's characteristics as being simply convenient assumptions to enable the project to proceed. However, the main problem arises with the question of empirical testing (5.2.3), since the answers to questions (5.3.1) and (5.3.2) seem to be "no", and this renders the other criteria (5.3.3)–(5.3.6) largely irrelevant.

Given the striking nature of Lenat's original claims (constructing a program which made "discoveries" comparable to those of Ramanujan, and which progressed from pre-numerical concepts to mathematics student knowledge in a few hours), it seems strange that no corroborative work has appeared in the past five years. It is one of the peculiarities of AI that, although replication of practical results is a cornerstone of traditional science, it is rare to see published accounts of repetitions of AI work. It is not clear how to interpret this phenomenon: it may be that few people have ever successfully re-implemented a large AI program, or it may be that those who do manage to repeat a published project (e.g. AM), do not regard this as publishable material. It may also be the case that an *unsuccessful* attempt at re-implementation would not be widely notified, since this might appear as an admission of incompetence. These circumstances impede the establishment of scientific standards within AI.

The positive contribution of the AM project is the proposal that mathematical discovery may be a systematic form of inference, based on a particular kind of knowledge-structure (hierarchically organised frames) and a particular arrangement of rules (indexed to the relevant concepts). What has *not* been achieved is a detailed, programmed test of these ideas, since the AM program departs from these original proposals. The negative contribution is that the impression has been given that such a test has been carried out (with impressive results), thus obscuring the real strengths and weaknesses of the scheme, and making it difficult for anyone to follow up this research systematically (cf. [11, pp. 8–9]). We believe that it would be extremely difficult to base further research in this area on AM, since the disparity betwen the written account and the actual program means that there is not in fact a tested theoretical basis from which to work. That is, we have a piece of work which publicly sets a very high standard of performance as the "current state of the art", but which gives insufficient information as to how one might, in practice, attain that performance.

There is a tendency for impressive AI programs to become part of the established "folklore" of the field, and to appear in introductory textbooks as exemplars of the achievements of AI. In view of the role of textbooks in passing on scientific knowledge to newcomers to the subject, it seems unfortunate that the picture presented should be based on unrepresentative "samples" of AI work.

Program-construction is the dominant research technique in AI, and this leads to the rather difficult problem of how the rest of the community can check a piece of work. In principle, each interested party could check the actual text of a program *and* its behaviour when running, and try to check the points (5.3.1)–(5.3.6) (particularly the correspondence to the proposed theory). Since this is rarely feasible for a complex program such as AM, much of this information remains private (i.e. known only by the creator of the program) rather than public, and it is not realistic to regard the mere existence of the program as remedying this state of affairs. Lenat has attempted (more than most other workers in AI) to render his program available to public assessment, both by making it available for running and by supplying such detailed appendices in his thesis. The whole discussion in this paper could not have commenced if Lenat had not provided this unusual level of documentation. If all work were as fully described, perhaps higher standards would develop naturally. What would be desirable would be a greater effort to supply theoretical statements (distinct from descriptions of implementations), a clearer (and, if possible, more publicly visible) correspondence between theory and program, and ways of reporting work in a detailed way which will be of direct practical use to sucessors in the field.

ACKNOWLEDGMENT

This work was carried out with the support of Science Research Council grant number GR/A/74760. We would like to thank Alan Bundy for his comments on an earlier draft.

REFERENCES

1. Bundy, A., Artificial mathematicians: the computational modelling of mathematical reasoning, Occasional Paper No. 24, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, 1980.
2. Bobrow, D.G. and Winograd, T., An overview of KRL, a knowledge representation language, *Cognitive Sci.* **1** (1) (1977) 3–46.
3. Fodor, J.A., Tom Swift and his procedural grandmother, *Cognition* **7** (1978) 229–247.
4. Hayes, P.J., Nine deadly sins, AISB European Newsletter (July 1975) 15–17.

5. Hempel, C., *Philosophy of Natural Science* (Prentice-Hall, Englewood Cliffs, NJ, 1966).
6. Hewitt, C., Viewing control structures as patterns of passing messages, *Artificial Intelligence* **8** (1977) 323–364.
7. Lenat, D.B., An artificial intelligence approach to discovery in mathematics as heuristic search, Memo AIM-286, Dept. of Computer Science, Stanford University, Stanford, CA, 1976.
8. Lenat, D.B., Automated theory formation in mathematics. Proc. Fifth International Joint Conference on Artificial Intelligence, MIT, Cambridge, MA (1977) 833–842.
9. Lenat, D.B., The ubiquity of discovery, Proc. Fifth International Joint Conference on Artificial Intelligence, MIT, Cambridge, MA (1977) 1093–1105.
10. Lenat, D.B., On automated scientific theory formation: a case study using the AM program, in: J.E. Hayes, D. Michie and O.I. Mikulich (Eds.), *Machine Intelligence* **9** (Ellis Horwood, Chichester, 1979) 251–283.
11. McDermott, D.V., Artificial intelligence meets natural stupidity, *SIGART Bulletin* **57** (1976) 4–9.
12. Meltzer, B., Review of [7], *AISB Quart.* **27** (1977) 20–23.
13. Minsky, M., A framework for representing knowledge, in: P. Winston (Ed.), *The Psychology of Computer Vision* (McGraw-Hill, New York, 1975) 211–277.
14. Newell, A., Harpy, production systems and human cognition, Rept. CMU-CS-78-140, Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1978.
15. Rudd, R.S., *Philosophy of Social Science* (Prentice-Hall, Englewood Cliffs, NJ, 1966).
16. Winograd, T., *Understanding Natural Language* (Edinburgh University Press, Edinburgh, 1972).

Received May 1981; revised version received October 1982