11

On Automated Scientific Theory Formation: A Case Study using the AM Program

D. B. Lenat†
Carnegie-Mellon University
Pittsburgh, USA

Abstract

A program called "AM" is described which carries on simple mathematics research, defining and studying new concepts under the guidance of a large body of heuristic rules. The 250 heuristics communicate via an agenda mechanism, a global priority queue of small tasks for the program to perform, and reasons why each task is plausible (for example, "Find generalizations of 'primes', because 'primes' turned out to be so useful a concept"). Each concept is represented as an active, structured knowledge module. One hundred very incomplete modules are initially supplied, each one corresponding to an elementary set-theoretic concept (for example, union). This provides a definite but immense space which AM begins to explore. In one hour, AM rediscovers hundreds of common concepts (including singleton sets, natural numbers, arithmetic) and theorems (for example, unique factorization). As AM defines concepts, and fills in their facets, it does not synthesize new heuristics for dealing effectively with those new concepts. This inability turns out to be its main limitation.

1. INTRODUCTION

1.1 Historical motivation

Scientists often face the difficult task of formulating nontrivial research problems which are soluble. In most branches of science, it is usually easier to tackle a specific given problem than to propose interesting yet manageable new questions to investigate. For example, contrast solving the Missionaries and Cannibals problem with the more ill-defined reasoning which led to inventing it. The first type of activity is formalizable and admits a deductive solution; the second is inductive and judgmental. As another example, contrast proving a given theorem and proposing it in the first place.

† Now at the Department of Computer Science, Stanford University, USA.

A wealth of AI research has been focused upon the former type of activity: deductive problem solving (see, for example, Bledsoe 1971, Nilsson 1971, Newell and Simon 1972). Approaches to inductive inference have also been made. Some researchers have tried to attack the problem in a completely domain-independent way (see, for example, Winston 1970). Other AI researchers believe that "expert knowledge" must be present if inductive reasoning is to be kept within the abilities of the human mind. Indeed, a few recent AI programs have incorporated such knowledge (in the form of judgmental rules gleaned from human experts) and successfully carried out quite complex inductive tasks: medical diagnosis (Shortliffe 1974), mass spectra identification (Feigenbaum 1971), clinical dialogue (Davis 1976), discovery of new mass spectroscopy rules (Buchanan 1975).

The next step in this progression of tasks would be that of fully automatic theory formation in some scientific field. This includes two activities: (i) discovering relationships among known concepts (for example, by formal manipulations, or by noticing regularities in empirical data), and (ii) defining new concepts for investigation. Meta-Dendral (Buchanan 1975) performs only the first of these (it doesn't develop new concepts); most domain-independent concept learning programs (Winston 1970) perform only the latter of these (while they do create new concepts, the initiative is not theirs but rather is that of a human "teacher" who already has the concepts in mind).

We are describing a computer program which defines new concepts, investigates them, notices regularities in the data about them, and conjectures relationships between them. This new information is used by the program to evaluate the newly-defined concepts, to concentrate upon the most interesting ones, and to iterate the entire process. This paper describes such a program: AM.

1.2 Choice of domain

Research in distinct fields of science and mathematics often proceeds slightly differently. Not only are the concepts different; so are most of the powerful heuristics. So it was reasonable that this first attempt should be limited to one narrow domain. Elementary mathematics was chosen, because:

1. There are no uncertainties in the raw data (arising, for example, from faulty measuring devices).
2. Reliance on experts' introspection is a powerful technique for codifying the judgmental rules needed to work effectively in a field. By choosing a familiar field, it was possible for the author to rely primarily on personal introspection for such heuristics.
3. The more formal a science is, the easier it is to automate (for example, the less one needs to use natural language to communicate information).
4. A mathematician has the freedom to explore - or to abandon - whatever he wants to. There is no specific problem to solve, no fixed "goal".
5. Unlike some fields (for example, propositional logic), elementary mathe-

matical research has an abundance (many hundreds) of powerful heuristic rules available.

6. One point of agreement between Weizenbaum and Lederberg (Buchanan *et al.*, 1976) is that AI can succeed in automating only those activities for which there exists a "strong theory" of how that activity is performed by human experts. AM is built on this kind of detailed model of mathematical research (see Sec. 1.3).

The limitations of mathematics as a domain are closely intertwined with its advantages. Having no ties to real-world data can be viewed as a liability, as can having no clear "right" or "wrong" behaviour. Since mathematics has been worked on for millenia by some of each culture's greatest minds, it is unlikely that a small effort like AM would make many startling new discoveries. Nevertheless, it was decided that the advantages outweighed the limitations, and the task domain of the program was settled.

1.3 Initial assumptions and hypotheses

The AM program got off the ground only because a number of sweeping assumptions were made about how mathematical research could be perfomed by a computer program:

1. Very little natural language processing capabilities are required. As it runs, AM is monitored by a human "user". AM keeps the user informed by instantiating English sentence templates. The user's input is rare and can be successfully stereotyped.
2. Formal reasoning (including proof) is not indispensable when doing theory formation in elementary mathematics. In the same spirit, we need not worry in advance about the occurence of contradictions.
3. Each mathematical concept can be represented as a list of facets (aspects, slots, parts, property/value pairs). For each new piece of knowledge gained, there will be no trouble in finding which facet of which concept it should be stored in.
4. The basic activity is to choose some facet of some concept, and then try to fill in new entries to store there; this will occasionally cause new concepts to be defined. The high-level decision about which facet of which concept to work on next can be handled by maintaining an ordered agenda of such mini-research tasks. The techniques for actually carrying out a task are contained within a large collection of heuristics.
5. Each heuristic has a well-defined domain of applicability, which coincides perfectly with one of AM's concepts. We can thus say the heuristic "belongs to" that concept.
6. Heuristics superimpose; they never interact strongly with each other. If one concept C1 is a specialization of concept C2, then C1's heuristics are more specific and more powerful, hence they should be tried first.
7. Each task (on the agenda of facet/concept tasks to be carried out) is

supported by a list of symbolic reasons, from which its priority is computed. We assume that the reasons always superimpose perfectly. They never change with time, and it makes no difference in what order they were noticed. It suffices to have a single, positive number for each reason, which characterizes its overall value.

8. The tasks on the agenda are completely independent. No task "wakes up" another. Only the general position (near the top, near the bottom) is of any significance (not the precise numeric value of its priority rating).
9. The set of heuristics need not grow, as new concepts are discovered. All commonsense knowledge required is assumed to be already present within the initially-given body of heuristic rules.

It is worth repeating that all the above points are merely convenient falsehoods. Their combined presence made AM do-able (by one person, in one year).

Point (4) above is a claim that a clean, simple model exists for mathematical research: a search process governed by a large collection of heuristic rules. Here is a simplified summary of that model:

1. The order in which a mathematics textbook presents a theory is almost the exact opposite of the order in which it was actually developed. In a text, definitions and lemmas are given with no motivation, and they turn out to be just the ones required for the next big theorem, whose proof magically follows. But in real life, a mathematician would (i) begin by examining some already-known concepts, (ii) try to find some regularity involving them, (iii) formulate those as conjectures to investigate further, and (iv) use them to motivate some simplifying new definitions.
2. Each of these four steps that the researcher takes involves choosing from a huge set of alternatives - that is, searching. He uses judgmental criteria (heuristics) to choose the "best" alternative. This saves his search from the combinatorial explosion.
3. Non-formal criteria (aesthetic interest, empirical induction, analogy, utility estimates) are much more important than formal methods, in the search for fruitful new definitions.
4. All such heuristics can be cast as situation/action (IF/THEN) rules. There is a common core of (a few hundred) heuristics, basic to all fields of mathematics at all levels. In addition to these, each field has several of its own rules; those are usually much more powerful than the general-purpose heuristics.
5. Nature is metaphysically pleasant: It is fair, uniform, regular. Statistical considerations are valid and valuable when trying to find regularity in mathematical data. Simplicity and synergy and symmetry abound.

1.4 Discovery in mathematics

By presenting a few examples, the preceding assumptions can, we hope, be made more plausible. We shall cite some scenarios of mathematical discoveries being

made. But before discussing how to synthesize a new mathematical theory, consider briefly how to analyse one, how to construct a plausible chain of reasoning which stretches from a given discovery all the way back to well-known concepts.

1.4.1 Analysis of a discovery

One can rationalize a given discovery by working backwards, by reducing the creative act to simpler and simpler creative acts. For example, consider the concept of prime numbers. How might one be led to define such a notion, if one had never heard of it before? Notice the following plausible strategy:

> If f is a function which transforms elements of A into elements of B, and B is ordered, then consider just those members of A which are transformed into extremal elements of B. This set is an interesting subset of A. Name it and study it.

When $f(x)$ means "divisors of x", and the ordering is "by length", this heuristic says "Consider those numbers which have a minimal number of factors - that is, the primes". So this rule actually reduces our task from "how in the world did somebody first think of the concept of 'prime numbers'?" to two more elementary problems: (i) "How might 'ordering-by-length' have been discovered?" and (ii) "How in the world did anybody first think of the concept of 'divisors of a number'?". The reduction was accomplished by citing the above heuristic. Bear in mind that it's just a rule of thumb, not a rule of inference. It can't guarantee anything, the way that Modus Ponens can guarantee to preserve validity. And yet, it is cost-effective for researchers to know and apply that heuristic rule, because (as in the above case) it frequently leads to valuable new discoveries.

Now suppose we know this general rule: "If f is an interesting relation, consider its inverse f^{-1}". It reduces the task of discovering divisors-of to the simpler task of discovering multiplication. Eventually, this task reduces to the discovery of very basic notions, like substitution, set-union, and equality. To explain how a given researcher might have made a given discovery, such an analysis could be continued until that inductive task had been reduced to "discovering" notions which the researcher already knew, which were his conceptual primitives.

1.4.2 Syntheses of discoveries

Suppose a large collection of these heuristic strategies has been assembled (for example, by analysing a great many discoveries, and writing down new heuristic rules whenever necessary). Instead of using them to explain how a given idea might have evolved, one can imagine starting from a basic core of knowledge and "running" the heuristics to generate new concepts. We're talking about reversing the process described in the last subsection: not how to explain discoveries, but how to make them.

Notice that this forward search is much "bushier", much more explosive, than was the backwards analysis previously described. Instead of having fixed

starting and ending concepts, we are now given only a starting point. This explains why it's much harder to actually make a discovery than to rationalize - by hindsight - how one might have made it. We have all noticed this phenomenon, the "Why-didn't-I-think-of-that-sooner!" feeling.

The forward search is quite explosive; we may hypothesize that the scientist employs some additional informal rules of thumb to constrain it. That is, he doesn't really follow rules like "Look at the inverse of each known relation f", because that would take up too much time. Rather, his heuristic rules might be more naturally stated as productions (condition/action rules) like this: "If a relation f is 1-1, and is very interesting, and Range(f) is much smaller than Domain(f), Then look at f^{-1}". Henceforth, "heuristic rule" will mean a conditional rule of thumb. In any particular situation, some subset of these rules will "trigger", and will suggest some relevant, plausible activities to perform. After following those suggestions, the situation will have changed, and the cycle will begin anew.

Such syntheses are precisely what the AM program - and perhaps what a human scientist - does. The program consists of a large corpus of primitive mathematical concepts, each with a few associated heuristics. Each such heuristic is a situation/action rule which functions as a local "plausible move generator". Some suggest tasks for the system to carry out, some suggest ways of satisfying a given task, etc. AM's activities all serve to expand AM itself, to enlarge upon a given body of mathematical knowledge. AM uses its heuristics as judgmental criteria to guide development in the most promising direction.

2. DESIGN OF THE 'AM' PROGRAM

A pure production system may be considered to consist of three components: data memory, a set of rules, and an interpreter. Since AM is more or less a rule-based system, it too can be considered as having three main design components: how it represents mathematical knowledge (its frame-like concept/facets scheme), how it enlarges its knowledge base (its collection of heuristic rules), and how it controls the firing of these rules (via the agenda mechanism). These form the subjects of the following three subsections.

2.1 Representation of concepts

The task of the AM program is to define plausible new mathematical concepts, and to investigate them. Each concept is represented internally as a bundle of slots or "facets". Each facet corresponds to some aspect of a concept, to some question we might want to ask about the concept. Since each concept is a mathematical entity, the kinds of questions one might ask are fairly constant from concept to concept. A set of 25 facets was therefore fixed once and for all. Below is that list of facets which a concept C may have. For each facet, we give a typical question about C which it answers.

Name: What shall we call C when talking with the user?
Generalizations: Which other concepts have less restrictive (that is, weaker) definitions than C?
Specializations: Which concepts satisfy C's definition plus some additional constraints?
Examples: What are some things that satisfy C's definition?
Isa's: Which concepts' definitions does C itself satisfy?
In-domain-of: Which operations can operate on C's.
In-range-of: Which operations result in C's when run?
Views: How can we view some X as if it were a C?
Intuitions: What abstract, analogic representations are known for C?
Analogies: Are there any similar concepts?
Conjec's: What are some potential theorems involving C?
Definitions: How can we tell if x is an example of C?
Algorithms: What exactly do we do to execute the operation C on a given argument?
Domain/Range: How many - and exactly what kinds of - arguments can operation C be executed on? What kinds of values will it return?
Worth: How valuable is C? (overall, aesthetic, utility, etc.)
Interest: What special features can make a C unusually interesting? Boring?

In addition, each facet F of concept C can possess a few little subfacets which contain heuristics for dealing with that facet of C's:

F.Fillin: What are some methods for finding new entries for facet F of a concept which is a C?
F.Check: How do we verify/debug potential entries for such a facet?
F.Suggest: If AM bogs down, what are some new tasks (related to facet F of concept C) to consider doing?

In the LISP implementation of AM, each concept is maintained as an atom with an attribute/value list (property list). Each facet, and its list of entries, is just a property and its associated value.

Below is a stylized rendition of the Sets concept, which intuitively corresponds to the notion of a collection of elements.

Name(s): Set, Proper Collection, Proper Class
Definitions:
Recursive: λ (S) [S={} or Set.Definition (Remove(Any-member(S),S))]
Recursive quick: λ (S) [S={} or Set.Definition (CDR(S))]
Quick: λ (S) [Match S with {...}]
Specializations: Empty-set, Nonempty-set, Singleton, Doubleton
Generalizations: Unordered-Structure, Collection,
Structure-with-no-multiple-elements-allowed
Examples:
Typical: {{}}, {A}, {A,B}, {3}
Barely: {}, {A, B,{C, {{{A, C, (3,3,9), ⟨4,{B},A⟩}}}}}
Not-quite: {A,A}, (), {B,A}
Foible: ⟨4,1,A,1⟩
Conjectures: All unordered-structures are sets.
Intuitions: Geometric: Venn diagram.
Analogies: {set, set operations}≡{list, list operations}
Worth: 600 [on a scale of 0-1000]
View:
Predicate: λ(P) {x∈Domain(P) | P(x)}
Structure: λ (S) Enclose-in-braces(Sort(Remove-multiple-elements(S)))
Suggest: If P is an interesting predicate over X,
Then consider {x∈X | P(x)}.
In-domain-of: Union, Intersection, Set-difference, Subset, Member,
Cartesian-product, Set-equality
In-range-of: Union, Intersect, Set-difference, Satisfying

To decipher the Definitions facet, there are a few things you must know. Facet F of concept C will occasionally be abbreviated as C.F. In those cases where F is "executable", the notion C.F will refer to applying the corresponding function. So the first entry in the Definitions facet is recursive because it contains an embedded call on the function Set.Definition. Notice that we are implying that the name of the lambda expression itself is "Set.Definition". Since there are three separate but equivalent definitions, AM may choose whichever one it wants when it recurs. AM can pick one via a random selection scheme, or always try to recur into the same definition as it was just in, or perhaps suit its choice to the form of the argument at the moment.

All concepts possess executable definitions (LISP predicates), though not necessarily effective ones. When given an argument x, Set.definition will return "True", "False", or will eventually be interrupted by a timer (indicating that no conclusion was reached about whether or not x is a set).

The "Views", "Intuitions", and "Analogies" facets must be distinguished from each other. "Views" is concerned with transformations between instances of two specific concepts (for example, how to view any predicate as a set, and vice versa). An entry on the "Analogies" facet is a mapping from a set of concepts (for example, between {bags, bag-union, bag-intersection, ...} and {numbers, addition, minimum, ...}; or between {primes, factoring, numbers, ...} and {simple groups, factoring into subgroups, groups ...}). "Intuitions" deals with transformations between a bunch of concepts and one of a few large, standard scenarios (for example, intuit the relation "≥" as playing on a see-saw; intuit a set by drawing a Venn diagram). Intuitions are characterized by being (i) opaque (AM cannot introspect on them, delve into their code), (ii) occasionally fallible, (iii) very quick, and (iv) carefully handcrafted in advance (since AM cannot pick up new intuitions via metaphors to the real world, as we humans can).

Since "Sets" is a static concept, it had no Algorithms facet (as did, for example, "Set-union"). The algorithms facet of a concept contains a list of entries, a list of equivalent algorithms. Each algorithm must have three separate parts:

1. Descriptors: Recursive, Linear, or Iterative? Quick or Slow? Opaque (difficult to analyse statically) or Transparent (cleanly coded)? Destructive or non-destructive?
2. Relators: Is this just a special case of some other concept's algorithm? Which others does this one call on? Is this similar to any other algorithms for any other concepts?
3. Program: A small, executable piece of LISP code. It may be used for actually "running" the algorithm; it may also be inspected, copied, reasoned about, etc.

There are multiple algorithms for the same concept because different ones have different properties: some are very quick in some cases, some are always slow but are very cleanly written and hence are easier to reason about, etc.

Another facet possessed only by active concepts is "Domain/Range". It is a list of entries, each of the form $<D_1\ D_2 \ldots D_i \rightarrow R>$, which means that the concept takes a list of arguments, the first one being an example of concept D_1, the second of D_2, ..., the last argument being an example of concept D_i, and if the algorithm (any entry on the Algorithms facet) is run on this argument list, then the value it returns will be an example of concept R. We may say that the Domain of the concept is the Cartesian product $D_1 \times D_2 \times \ldots \times D_i$, and that the Range of the concept is R. For example, the Domain/Range of Set-union is <Sets Sets → Sets>; Set-union takes a pair of sets as its argument list, and returns a set as its value.

Several other facets were considered from time to time, including "Uninterestingness", "Justification", "Recognition", etc. They were all dropped eventually, because of their insignificant contribution to the performance of the AM program. The Intuitions facet was eventually dropped, because it never led to any discoveries which had not been foreseen by the author.

Once the representation of knowledge is settled, there remains the actual choice of what knowledge to put into the program initially. One hundred elementary concepts were selected, corresponding roughly to what Piaget might have called "prenumerical knowledge". Appendix 1 presents a graph of these concepts, showing their interrelationships of Generalization/Specialization and Examples/Isa's. There is much static structural knowledge (sets, truth-values, conjectures ...) and much knowledge about simple activities (boolean relations, composition of relations, set operations, ...). Notice that there is no notion of proof, of formal reasoning, or of numbers or arithmetic.

2.2 Top-level control: the agenda

AM's basic activity is to find new entries for some facet of some concept. But which particular one should it choose to develop next? Initially, there are over one hundred concepts, each with about twenty blank facets; thus the "space" from which to choose is of size two thousand. As more concepts are defined, this number increases. It's worth having AM spend some time deciding which basic task (facet/concept) to work on next, for two reasons: most of the tasks will never be explored, and only a few of the tasks will appear (to the human user) rational things to work on at the moment.

Much informal expert knowledge is required to constrain the search, to quickly zero in on one of these few very good tasks to tackle next. This is done in two stages:

1. A list of plausible facet/concept pairs is maintained. No task can get onto this "agenda" unless there is some reason why working on that facet of that concept would be worthwhile.
2. Each task on this agenda is assigned a priority rating, based on the number (and strengths) of reasons supporting it. This allows the entire agenda to be kept ordered by plausibility.

The first of these constraints is much like replacing a legal move generator with a plausible move generator, in a heuristic search program. The second kind of constraint is akin to using a heuristic evaluation function to select the best move from among the good ones. Here is a typical entry on the agenda, a task:

Activity:	Fill in some entries
Facet:	for the GENERALIZATIONS facet
Concept:	of the PRIMES concept
Reasons:	because
(1)	There is only 1 known gen'l. of Primes, so far.
(2)	The worth rating of Primes is now very high.
(3)	Focus of attention: AM just worked on Primes.
(4)	Very few numbers are primes; a slightly more plentiful concept may be more interesting.
Priority:	350 [on a scale of 0-1000]

The actual top-level control policy is to pluck the top task (highest priority rating) from the agenda, and then execute it. While a task executes, some new tasks may be proposed (and merged into the agenda), some new concepts may get created, and (one hopes) some entries for the specified facet of the specified concept will be found and filled in. Once a task is chosen, the priority rating of that task then serves a new function: it is taken as an estimate of how much computational resource to devote to working on this task. The task above, in the box, might be allotted 35 cpu seconds and 350 list cells, because its rating was 350. When either resource is exhausted, work on the task halts. The task is removed from the agenda, and the cycle begins anew (AM starts working on whichever task is now at the top of the agenda).

2.3 Low-level control: the heuristics

After a task is selected from the agenda, how is it "executed"? A concise answer would be: AM selects relevant heuristics and executes them; they satisfy the task via side-effects. This really just splits our original question into two new ones: How are relevant heuristics located? What does it mean for a heuristic to be executed and to achieve something?

2.3.1 How relevant heuristics are located

Each heuristic is represented as a condition/action rule. The condition or left-hand side of a rule tests to see whether the rule is applicable to the task on hand. The action or right-hand side of the rule consists of a list of actions to perform if the rule is applicable. Below is a typical heuristic:

IF the current task is to check examples of a concept X,
and (Forsome Y) Y is a generalization of X,
and Y has at least 10 known examples
and all examples of Y are also examples of X,

THEN conjecture: X is really no more specialized than Y,
and add that conjecture as a new entry on the
Examples facet of the Conjecs concept,
and add the following task to the agenda:
"Check examples of Y"
for this reason: Y may analogously turn out to be
equal to one of its supposed generalizations.

It is the heuristics' right-hand (THEN-) sides which actually accomplish the selected task; that process will be described in the next subsection. The left-hand (IF-) sides are the relevancy checkers, and will be focused on now:

Syntactically, the left side must be a predicate, a LISP function which returns True or False depending upon the situation at that moment. It must be a conjunction $P_1 \wedge P_2 \wedge P_3 \wedge \ldots$ of smaller predicates P_i, each of which must be quick and must have no side effects. Here are five typical conjuncts which might appear within rules' left-hand sides:

Over half of the current task's time allotment is used up;
There are some known examples of Structures;
Some known generalization of the current concept (the concept mentioned as part of the current task) has a completely empty Examples facet;
A task recently worked on had the form "Fill in facet F of C", for any F, where C is the current concept;
The user has used this program at least once before;

It turned out that the laxity of constraints on the form of the heuristic rules proved excessive: it made it very difficult for AM to analyse and modify its own heuristics.

From a "pure production system" viewpoint, we have answered the question of locating relevant heuristics. Namely, we evaluate the left sides of all the rules, and see which ones respond "True". But AM contains hundreds of heuristics, and to repeatedly evaluate each one's condition would use up tremendous amounts of time. AM is able quickly to select a set of potentially relevant rules, rules whose left sides are then evaluated to test for true relevance. The secret is that each rule is stored somewhere a propos to its "domain of applicability". The proper place to store the rule is determined by the first conjunct on its left-hand side. Consider this heuristic:

IF the current task is to find examples of activity F,
and a fast algorithm A for computing F is known,
THEN one way to get examples of F is to run A on
randomly chosen examples of the Domain of F.

The very first conjunct of a rule's left side is always special. It specifies the domain of applicability (potential relevance) of the heuristic, by naming a particular facet of a particular concept to which this rule is relevant (in the above rule, the domain of relevance is therefore the Examples facet of the Activity concept). AM uses such first conjuncts as pre-preconditions: Each potentially relevant rule can be located by its first conjunct alone. Then, its left-hand side is fully evaluated, to indicate whether it's truly relevant. Here are a few typical expressions which could be first conjuncts:

The current task (the one just selected from the agenda) is of the form "Check the Domain/range facet of concept X", where X is some surjective function;
The current task matches "Fill in boundary examples of X", where X is an operation on pairs of sets;
The current task is "Fill in examples of Primes";

The key observation is that a heuristic typically applies to all examples of a particular concept C. The rule above has C = Activity; it's relevant to each individual activity. For example, it can be used to find examples of Set-union, since Set-union is an activity.

When a task is chosen, it specifies which concept C and which facet F are to be worked on. AM then "ripples upward" to gather potentially relevant rules: it looks on facet F of concept C to see if any rules are tacked on there, it looks on facet F of each generalization of C, on each of their generalizations, etc. If the current task were "Check the Domain/range or Union-o-Union",† then AM would ripple upward from Union-o-Union, along the Generalization facet entries, gathering heuristics as it went. The program would ascertain which concepts claim Union-o-Union as one of their examples. These concepts happen to include Compose-with-self, Compose, Operation, Active, Any-concept, Anything. AM would collect heuristics that tell how to check the Domain/range of any composition, how to check the Domain/range facet of any concept, etc. Of course, the further out it ripples, the more general (and hence weaker) the heuristics tend to be. Here is one heuristic, tacked onto the Domain/range facet of Operation, which would be garnered if the selected task were "Check Domain/range of Union-o-Union":

IF the current task is "Check the Domain/range of F", for some Activity F,
and an entry on that facet has the form ⟨D D...D→ R⟩,
and concept R is a generalization of concept D,
THEN it is worth spending time checking whether or not
the range of F might be simply D, instead of R.

Suppose that one entry on Union-o-Union's Domain/range facet was "<Nonempty-sets Nonempty-sets Nonempty-sets → Sets>". Then the above heuristic would be truly relevant (all three conjuncts on its left-hand side would be satisfied), and it would pose the question: Is the union of three nonempty sets always nonempty? Empirical evidence would eventually confirm this, and the Domain/range facet of Union-o-Union would then contain that fact. AM would ask the same question for the operation Intersect. Although the answer in that case is negative, it is nonetheless a rational idea to investigate whether or not the intersection of two nonempty sets is always nonempty.

Here is another way to look at the heuristic-gathering process. All the concepts known to AM are arranged in a big hierarchy, via subset-of links (Specializations and Generalizations) and element-of links (Isa's and Examples) as diagrammed in Appendix 1. Since each heuristic is associated with one individual concept (its domain of applicability), there is a hierarchy induced

†This operation is the result of composing set-union with itself. It performs $\lambda(x,y,z)\, x \cup (y \cup z)$.

upon the set of heuristics. Heritability properties hold: a heuristic tacked onto concept C is applicable to working on all "lower" (more specialized) concepts. This allows us efficiently to analogically access the potentially relevant heuristics simply by chasing upward links in the hierarchy. Note that the task selected from the agenda provides an explicit pointer to the "lowest" - most specific - concept; AM ripples upward from it. Thus concepts are gathered in order of increasing generality; hence so are the heuristics.

Below are summarized the three main points that comprise AM's scheme for finding relevant heuristics in a "natural" way and then using them:

1. Each heuristic is tacked onto the most general concept for which it applies: it is given as large a domain of applicability as possible. This will maximize its generality, while leaving its power untouched, hence bringing it as close as possible to the ideal tradeoff between generality and power.
2. When the current task deals with concept C, AM ripples upward from C, tracing along Generalization and Isa links, to quickly find all concepts which claim C as one of their examples. Heuristics attached to all such concepts are potentially relevant.
3. All heuristics are represented as condition/action rules. As the potentially relevant rules are located (in step 2), AM evaluates each's left-hand side, in order of increasing generality. The rippling process automatically gathers the heuristics in this order. Whenever a rule's left side returns True, the rule is known to be truly relevant, and its right side is immediately executed.

2.3.2 What happens when heuristics are executed

When a rule is recognized as relevant, its right side is executed. Precisely how does this accomplish the chosen task?

The right side, by contrast to the left, may take a great deal of time, have many side effects, and return a value which is simply ignored. The right side of a rule is a series of little LISP functions, each of which is called an *action.* Semantically, each action performs some processing which is appropriate in some way to the kinds of situation in which the rule's left side would have been satisfied (returned True). The only constraint which each action must satisfy is that it have one of the following three kinds of side-effects, and no other kinds:

1. It suggests a new task to add to the agenda.
2. it dictates how some new concept is to be defined.
3. It adds some entry to some facet of some concept.

Bear in mind that the right side of a single rule is a list of such actions. Let's now treat these three kinds of actions:

2.3.2.1 Heuristics suggest new tasks

The left side of a rule triggers. Scattered among the list of "things to do" on its right side are some suggestions for future tasks. These new tasks are then simply

added to the agenda. The suggestion for the task includes enough information about the task to make it easy for AM to assemble its parts, to find reasons for it, numerically to evaluate those reasons, etc. For example, here is a typical rule which proposes a new task. It says to generalize a predicate if it appears to be returning "True" very rarely:

IF the current task was "Fill in examples of X", for some predicate X,
    and over 100 items are known in the domain of X,
    and at least 10 cpu secs. have been spent so far on this task,
    and X has returned True at least once,
    and X returned False over 20 times as often as True,
THEN add the following task to the agenda:
    "Fill in generalizations of X"
    for the following reason:
    "X is rarely satisfied; a slightly less restrictive
        predicate might be much more interesting"
    This reason has a rating which is the False/True results ratio

Let's see one instance where this rule was used. AM worked on the task "Fill in examples of List-Equality". One heuristic (displayed in Sec. 2.3.1, and again in detail in Sec. 2.3.2.3) said: randomly pick elements from that predicate's domain and simply run the predicate. Thus AM repeatedly plucked random pairs of lists, and tested whether or not they were equal. Needless to say, not a high percentage returned True (in practice, 2 out of 242). This rule's left side was satisfied, and it executed. Its right side caused a new task to be formulated: "Fill in generalizations of List-Equality". The reason was as stated above in the rule, and that reason got a numeric rating of 240/2 = 120. That task was then assigned an overall rating (in this case, just 120) and merged into the agenda. It sandwiched in between a task with a rating of 128 and one with a 104 priority rating. Incidentally, when this task was finally selected, it led to the creation of several interesting concepts, including the predicate which we might call "Same-length".

2.3.2.2 Heuristics create new concepts

One of the three kinds of allowable actions on the right side of a heuristic rule is to create a specific new concept. For each such creation, the heuristic must specify how the new concept is to be constructed. The heuristic states the Definition facet entries for the new concept, plus usually a few other facets' contents. After this action terminates, the new concept will "exist". A few of its facets will be filled in, and many others will be blank. Some new tasks may be added to the agenda at concept-time, tasks which indicate that AM ought to spend some time filling in some of those blank facets in the near future. Here is a heuristic rule which results in a new concept being created:

IF the current rask was "Fill in examples of F"
        for an operation F, say from domain A into range B,
    and more than 100 items are known examples of A,
    and more than 10 range items (examples of B) were
        found by applying F to these domain elements,

and at least one of these range items 'b' is a distinguished
member (especially, an extremum) of B,
THEN for each such 'b'∈B, create the following kind of concept:

NAME: F^{-1}-of-b
DEFINITION: λ (a) F(a) is a 'b'
GENERALIZATIONS: A
WORTH: Average(Worth(A), Worth(B), Worth(b), Worth(F), ||Examples(B)||)
INTEREST: Any conjec. involving this concept and either F or F^{-1}

and the reason for this creation is:
"It's worth investigating A's which have unusual F-values"
and add five new tasks to the agenda,
each of the form "Fill in facet x of F^{-1}-of-b"
where x is Conjectures, Gereralizations, Specializations, Examples, Isa's;
each for the following reason:
"This concept was newly synthesized; it is crucial
to find where it 'fits in' to the hierarchy"
The reason's rating is computed as:
Worth(F^{-1}-of-b) = Arg(Worth(F), Worth(b)).

One use of this heuristic was when the current task was "Fill in examples of Divisors-of". The heuristic's left side was satisfied because: Divisors-of is an operation (from Numbers to Sets of numbers), and far more than the required 100 different numbers are known, and more than 10 different sets of factors were located altogether, and some of them were in fact distinguished by being extreme kinds of sets (for example, singletons, empty sets, doubletons, tripletons, ...). After its left side triggered, the right side of the rule was executed. Four new concepts were created immediately. Here is one of them:

NAME: Divisors-of^{-1}-of-Doubleton
DEFINITION: λ (a) Divisors-of(a) is a Doubleton
GENERALIZATIONS: Numbers
WORTH: 100
INTEREST: Any conjec. involving this concept and either Divisors-of or Times

This is a concept representing a certain class of numbers, in fact the numbers we call "primes". The heuristic rule is of course applicable to any kind of operation, not just numeric ones. As another instance of its use, consider what happened when the current task was "Fill in examples of Set-intersect". This rule caused AM to notice that some pairs of sets were mapping over into the most extreme of all sets: the empty set. The rule then had AM define the new concept we would call "disjointness": pairs of sets having empty intersection. Similarly, "subset" arose as the relation that holds between sets A and B iff Set-difference$(A,B)=\{\}$. So we see how the above heuristic rule led to the discovery of many well-known concepts.

Here is just a tiny bit of "theory" behind how these concept-creating rules were designed: A facet of a neonatal concept is filled in immediately at birth iff both (i) it's trivial to fill in at creation-time, and (ii) it would be very difficult to fill in later on. The following facets are typically filled in right away: Definitions, Algorithms, Domain/range, Worth, plus a pointer to a "parent" concept (for example, the trivially-computed entry "Numbers" for the Generalizations facet of the Primes concept). Each other facet is either left unmentioned by the rule,

or else is explicitly made the subject of a new task which gets added to the agenda. For instance, the heuristic rule above would propose five new tasks at the moment that the Primes concept was created, including "Fill in conjectures about Primes", "Fill in specializations of Primes", "Fill in examples of Primes", etc.

2.3.2.3 Heuristics fill in entries for a specific facet

If the task plucked from the agenda were "Fill in examples of Set-union", it would not be too much to hope for that by the time all the heuristic rules had finished executing, some examples of that operation would indeed exist on the Examples facet of the Set-union concept. Let's see how this can happen.

AM starts by rippling upward from Set-union, looking for heuristics which are relevant to finding examples of Set-union (there are no such rules), relevant to finding examples of Set-operations, of Operations, of any Activity, of any Concept, of Anything. Here is one rule garnered in the search, a rule which is tacked onto (hence assumed applicable to) the Examples facet of Activity:

IF the current task is to fill in examples of activity F,
    and there is a fast known algorithm for F,
THEN one way to get examples of F is to run F's algorithm
    on randomly chosen examples of the domain of F.

Of course, in the LISP implementation, this situation-action rule is not coded quite so neatly. It would be more faithfully translated as follows:

IF CURR-TASK matches (FILLIN EXAMPLES F←any-activity),
    and the Algorithms facet of F contains an entry with descriptor "Quick",
THEN carry out the following procedure:
    1. Find the domain of F, and call it D;
    2. Find examples of D, and call them E;
    3. Find a fast algorithm to compute F; call it A;
    4. Repeatedly:
        4a. Choose any member of E, and call it E1.
        4b. Run A on E1, and call the result X.
        4c. Check whether ⟨E1,X⟩ satisfies the definition of F.
        4d. If so, then add ⟨E1 → X⟩ to the Examples facet of F.
        4e. If not, then add ⟨E1 → X⟩ to the Non-examples facet of F.

Let's see exactly how this rule found examples of Set-union. Step (1) says to locate the domain of Set-union. The facet labelled Domain/range, on the Set-union concept, contains the entry (SET SET → SET), which indicates that the domain is a pair of sets. That is, Set-union is an operation which accepts (as its arguments) two sets.

Since the domain elements are sets, step (2) says to locate examples of sets. The facet labelled Examples, on the Sets concept, points to a list of about 30 different sets. This includes {*Z*}, {*A*,*B*,*C*,*D*,*E*}, {}, {*A*,{{*B*}}},...

Step (3) involves nothing more than accessing some entry tagged with the descriptor "Quick" on the Algorithms facet of Set-union. One such entry is a recursive LISP function of two arguments, which halts, when the first argument is the empty set, and otherwise pulls an element out of that set, Set-inserts it

into the second argument, and then recurs on the new values of the two sets. For convenience, we'll refer to this algorithm as UNION.

We then enter the loop of Step (4). Step(a) has us choose one pair of our examples of sets, say the first two {*Z*} and {*A,B,C,D,E*}. Step (4b) has us run UNION on these two sets. The result is {*A,B,C,D,E,Z*}. Step (4c) has us grab an entry from the Definitions facet of Set-union, and run it. A typical definition is this formal one:

(λ (*S*1 *S*2 *S*3)
  (AND
    (For all *x* in *S*1, *x* is in *S*3)
    (For all *x* in *S*2, *x* is in *S*3)
    (For all *x* in *S*3, *x* is in *S*1 or *x* is in *S*2))).

It is run on the three arguments *S*1={*Z*}, *S*2={*A,B,C,D,E*}, *S*3={*A,B,C,D,E,Z*}. Since it returns "True", we proceed to Step (4d). The construct ⟨{*Z*}, {*A,B,C,D,E*} → {*A,B,C,D,E,Z*}⟩ is added to the Examples facet of Set-union.

At this stage, control returns to the beginning of the Step (4) loop. A new pair of sets is chosen, and so on.

But when would this loop stop? Recall that as soon as a task is selected from the agenda, it is assigned a time and a space allotment (based on its priority value). If there are many different rules all claiming to be relevant to the current task, then each one is allocated a small fraction of those time/space quanta. When either of these resources is exhausted, AM would break away at a "clean" point (just after finishing a cycle of the Step (4) loop) and would move on to a new heuristic rule for filling in examples of Set-union.

3. RESULTS

3.1 Excerpt of the AM program running

Repeatedly, the top task is plucked from the agenda, and heuristics are executed in an attempt to satisfy it. AM has a modest facility that prints out a description of these activities as they occur. Below is a tiny excerpt of this self-trace monologue, in which AM discovers prime numbers. In Appendix 3, the reader may observe (in much more condensed form) summaries of the tasks which preceded these, tasks in which elementary set theory was explored, cardinality was discovered, and arithmetic was developed.

** *TASK CHOSEN:* ** Fill in Examples of the concept "Divisors-of".
  3 Reasons:
    (1) No known examples of Divisors-of yet.
    (2) Times (related to Divisors-of) is now very interesting.
    (3) Focus of attention: AM just defined Divisors-of.
26 examples found, in 9 secs. e.g., Divisors-of(6)={1,2,3,6}

** *TASK CHOSEN:* ** Consider numbers having small sets of Divisors-of.
2 Reasons:
(1) Worthwhile to look for extreme cases.
(2) Focus of attention: AM just worked on Divisors-of.
Filling in examples of numbers with 0 divisors.
0 examples found, in 4.0 seconds.
Conjecture: no numbers have precisely 0 divisors.
Filling in examples of numbers with 1 divisors.
1 examples found, in 4 secs. e.g., Divisors-of(1)={1}.
Conjecture: 1 is the only number with exactly 1 divisor.
Filling in examples of numbers with 2 divisors.
24 examples found, in 4 secs. e.g., Divisors-of(13)={1,13}.
No obvious conjecture. May merit more study.
Creating a new concept: "Numbers-with-2-divisors".
Filling in examples of numbers with 3 divisors.
11 examples found, in 4 secs. Divisors-of(49)={1,7,49}.
All numbers with 3 divisors are also Perfect Squares. Unexpected!.
The chance of coincidence is below acceptable limits.
Creating a new concept: "Numbers-with-3-divisors".

** *TASK CHOSEN:* ** Consider square-roots of Numbers-with-3-divisors.
2 Reasons:
(1) All known Numbers-with-3-divisors unexpectedly turned out to all be Perfect Squares as well.
(2) Focus of attention: AM just defined Numbers-with-3-divisors.
All square-roots of Numbers-with-3-divisors seem to be Numbers-with-2-divisors.
E.g., Divisors(169) = Divisors(13) = {1,13}.
Even the converse of this seems empirically to be true.
I.e., the square of each Number-with-2-divisors seems to be a Number-with-3-divisors.
The chance of coincidence is below acceptable limits.
Boosting the Worth rating of both concepts.

** *TASK CHOSEN:* ** Consider the squares of Numbers-with-3-divisors.
3 Reasons:
(1) Squares of Numbers-with-2-divisors were very interesting.
(2) Square-roots of Numbers-with-3-divisors were interesting.
(3) Focus of attention: AM just worked on Numbers-with-3-divisors.

The last task goes nowhere, and is a good place to terminate this excerpt and this subsection.

3.2 Overall performance

AM began its investigations with scanty knowledge of a hundred elementary concepts of finite set theory (Appendix 1). Most of the obvious finite set-theoretic

concepts and relationships were quickly found (for example, de Morgan's laws; singletons), but no sophisticated set theory was ever done (for example, diagonalization).

Rather, AM decided that "equality" was worth generalizing, and thereby discovered the relation "same-size-as". "Natural numbers" were based on this, and soon most simple arithmetic operations were defined (as analogs to set-theoretic operations; for example, "subtract" is the analog of "Set-difference"). See Appendix 2.

Since addition arose as an analog to union, and multiplication as a repeated substitution, it came as quite a surprise to AM when it noticed that they were related (namely, $N+N=2\times N$). AM later rediscovered multiplication in three other ways: as repeated addition, as the numeric analog of the Cartesian product of sets, and by studying the cardinality of power sets. These operations were defined in different ways, so it was an unexpected (to AM) discovery when they all turned out to be equivallent. These surprises caused AM to give the concept 'Times' quite a high Worth rating. Exponentiation was defined as repeated multiplication. Unfortunately, AM never found any obvious properties of exponentiation, hence lost all interest in it.

Soon after defining multiplication, AM investigated the process of multiplying a number by itself: squaring. The inverse of this turned out to be interesting, and led to the definition of square-root. AM remained content to play around with the concept of integer-square-root. Although it isolated the set of numbers which had no square root, AM was never close to discovering negative numbers, let alone irrationals. No notion of "closure" was provided to - or discovered by - AM.

Raising to fourth-powers, and fourth-rooting, were discovered at this time. Perfect squares and perfect fourth-powers were isolated. Many other numeric operations and kinds of numbers were found to be of interest: Odds, Evens, Doubling, Halving, etc. Primitive notions of numeric inequality were defined, but AM never even discovered Trichotomy.

The associativity and commutativity of multiplication indicated that it could accept a Bag of numbers as its argument. When AM defined the inverse relation corresponding to Times, this property allowed the definition to be: "λ (x) any bag of numbers (each >1) whose product is x". This was just the notion of factoring a number x. Minimally-factorable numbers turned out to be what we call primes. Maximally-factorable numbers were also thought to be interesting, and this motivated some new results in number theory (see Lenat 1976, Appendix 4).

Prime pairs were discovered in a bizarre way - by restricting the domain and range of addition to primes (that is, solutions of $p+q=r$ in primes).

AM conjectured the fundamental theorem of arithmetic (unique factorization into primes) and Goldbach's conjecture (every even number >2 is the sum of two primes) in a surprisingly symmetric way. The unary representation of numbers gave way to a representation as a bag of primes (based on unique

factorization), but AM never thought of exponential notation. Diophantine equations were isolated, but not developed very far.

Since the key concepts of remainder, greater-than, gcd, and exponentiation were never mastered, progress in number theory was arrested. Other crucial concepts which were never uncovered include: infinity, proof, rationals, residues, etc. Most of these "omissions", could have been discovered by the existing heuristic rules in AM. The paths which would have resulted in their definition were simply never rated high enough (by AM, by itself) to warrant exploration.

All the discoveries mentioned (including all those in Appendix 2) were made in a run lasting one cpu hour (Interlisp + 100K, Sumex PDP-10 KI). Two hundred jobs in toto were selected from the agenda and executed; many of these are summarized (in order) in Appendix 3. On the average, a job was granted 30 cpu seconds, but actually used only 18 seconds (by which time all the truly relevant heuristics had finished executing). For a typical job, about 35 rules were located as potentially relevant, and about a dozen actually fired (were executed). AM began with 115 concepts and ended up with three times that many. Of the synthesized concepts, half were technically termed "losers" (both by the author and by AM; for example, "Subtract-x-from-itself" was a real zero), and half the other new concepts were only marginal (for example, "Numbers which are uniquely representable as the sum of two primes"). Two hundred and fifty heuristic rules were present during this run of the program.

Although AM fared well according to several different measures of performance (see Sec. 3.4), its limitations have considerable significance. As AM ran longer and longer, the concepts it defined were further and further from the primitives it began with. For example, "prime-pairs" were defined using "primes" and "addition", the former of which was defined from "divisors-of", which in turn came from "multiplication", which arose from "addition", which was defined as a restriction of "union", which (finally!) was a primitive concept that we had supplied (with heuristics) to AM initially. When AM subsequently needed help with prime pairs, it was forced to rely on rules of thumb supplied originally about unioning. Although the heritability property of heuristics did ensure that those rules were still valid, the trouble was that they were too general, too weak to deal effectively with the specialized notions of primes and arithmetic.

For instance, one general rule indicated that $A \cup B$ would be interesting if it possessed properties absent both from A and from B. This translated into the prime-pair case as "If $p+q=r$, and p,q,r are primes, Then r is interesting if it has properties not possessed by p or by q". The search for categories of such interesting primes r was of course barren. It showed a fundamental lack of understanding about numbers, addition, odd/even-ness, and primes. As another example, AM didn't recognize a priori that the UFT (unique factorization theorem) was more significant than Goldbach's conjecture.

The key deficiency was the lack of adequate meta-rules (Davis 1976): heuristics which reason about heuristics: keep track of their performance,

modify them, create new ones, etc. Here is one such rule, which would have taken care of the "Goldbach vs UFT" problem:

> After applying the "look at the inverse of extrema" heuristic, and thereby defining a new concept C (as f^{-1} of b), where C is a new specialization of concept A,
> Synthesize a heuristic which indicates that conjectures involving C and f (or f^{-1}) are very significant and natural, whereas those involving C and unrelated operations are probably anomalies,
> and synthesize another heuristic which indicates that C is a good kind of A upon which to test conjectures involving f or f^{-1}.

How would this meta-rule be used? When primes are defined as the inverse image of doubletons, under the operation "divisors-of", the meta-rule would trigger, and two brand new rules would be synthesized. The first of those new heuristics would say that conjectures about primes were natural iff they involved multiplication or division. Thus the UFT would be rated as important, and Goldbach's conjecture as cute but useless. The second new rule would say that Primes are a useful kind of Number upon which to test out conjectures involving multiplication or division; this, too is quite a powerful piece of informal knowledge.

Aside from the preceding major limitation, most of the other problems pertain to missing knowledge: Many concepts one might consider basic to discovery in mathematics are absent from AM; analogies were under-utilized; physical intuition was hand-crafted only; the interface to the user was far from ideal; etc.

3.3 Experiments with AM

One valuable aspect of AM is that it is amenable to many kinds of experiment. Although AM is too ad hoc for numeric results to have much significance, the qualitative results of such experiments may have some valid implications for mathematical research, for automating mathematical research, and for designing "scientist assistant" programs.

3.3.1 Must the Worth numbers be finely tuned?

To signify its overall worth, each of the 115 initial concepts had a rating number (0-1000) supplied by the author. The worth ratings affect the overall priority values of tasks on the agenda. Just how sensitive is AM's behaviour to the initial settings of the Worth numbers?

To test this, a simple experiment was performed. All the concepts' Worth facets were set to 200 initially. By and large, the same discoveries were made as before. But there were now long periods of blind wandering (especially near the beginning of the run). Once AM hooked into a line of productive developments, it advanced at the old rate. During such chains of discoveries, AM was guided by massive quantities of symbolic reasons for the tasks it chose, not by nuances in numeric ratings. As these spurts of development died out, AM would wander around again until the next one started.

3.3.2 How finely tuned is the agenda?

The top few tasks on the agenda almost always appear to be reasonable things to do at the time. But what if, instead of picking the top-rated task, AM is always made to select one randomly from the top 20 tasks on the agenda? In that case, AM's rate of discovery is slowed only by about a factor of 3. But the apparent "rationality" of the program (as perceived by a human onlooker) disintegrates.

3.3.3 How valuable is the presence of symbolic 'reasons'?

One effect of note was observed: When a task is proposed which already exists on the agenda, then it matters very much whether the task is being suggested for a new reason or not. If the reason is an old, already-known one, then the priority of the task on the agenda shouldn't rise very much. But if it is a brand new reason, then the task's rating should be boosted tremendously. The importance of this effect argues strongly in favour of having symbolic justification of the rank of each task on a priority queue, not just "summarizing" each task's set of reasons by a single number.

3.3.4 What if certain concepts are excised?

As expected, eliminating certain concepts did seal off whole sets of discoveries to the system. For example, excising Equality prevented AM from discovering Cardinality. One surprising result was that many common concepts get discovered in several ways. For instance, multiplication arose in no fewer than four separate chains of discoveries.

3.3.5 Can AM work in the new domain of plane geometry?

One demonstration of AM's generality (for example, that its "Activity" heuristics really do apply to any activity) would be to choose some new mathematical field, add some concepts from that domain, and then let AM loose to discover new things. Only one experiment of this type was actually carried out on the AM program.

Twenty concepts from elementary plane geometry were defined for AM (including Point, Line, Angel, Triangle, Equality of points/lines/angles/triangles). No new heuristics were added to AM.

AM was able to find examples of all the supplied concepts, and to use the character of such empirical data to determine reasonable directions to proceed in its search. AM derived the concepts of congruence and similarity of triangles, plus many other well-known concepts. An unusual result was the repeated derivation of the concept of "timberline": this is a predicate on two triangles, which is true iff they share a common vertex and angle, and if their opposite sides are parallel. AM also came up with a cute geometric interpretation of Goldbach's conjecture: Any angle (0–180°) can be approximated to within 1° as the sum of two angles each of a prime number of degrees. But lacking a geometry "model" (an analogic representation of Euclidean space, for example, the kind that Gelernter (1963) employed,) AM was doomed to propose many implausible geometric conjectures.

More and more drastic changes to the knowledge base of AM would be required if its task domain were to be shifted further and further from simple finite set theory. For example, to work reasonably well in analysis, AM would need the additional concepts of continuity, infinity, limits, measure, etc. To work in a non-formalized field (such as mass spectroscopy), AM would have to be spoon-fed data, the way that Meta-Dendral is, or else be connected directly to physical sensing devices to that it could gather its own empirical data. One other alternative would be to provide AM with a formal model of some real-world phenomenon (for example, gravitation). But in such a case, AM could never do more than reformulate the model, could never discover any "real" effects which were not taken into account by the model. If fed a model of a Newtonian world, AM might discover Lagrangian mechanics, but it could never observe any relativistic effects. The impracticality of all these alternatives seems to indicate that AM-like programs are best suited to theory formation in fully formalizable fields (mathematics, programming, games, etc.).

3.4 Evaluating the AM program

We may wish to evaluate AM by using various criteria. Some obvious ones, with capsule results, appear below:

1. *By AM's ultimate achievements.* Besides discovering many well-known useful concepts, AM discovered some which aren't widely known: maximally-divisible numbers, numbers which can be uniquely represented as the sum of two primes, timberline. The first of these is related to Ramanujan's "highly composite numbers", and represents a real (albeit miniscule) contribution to number theory.

2. *By the character of the differences between initial and final states.* AM moved all the way from finite set theory to divisibility theory, from sets to numbers to interesting kinds of numbers, from skeletal concepts (none of which had any Examples filled in) to completed concepts, from one hundred concepts to three hundred.

3. *By the quality of the route AM took to accomplish this mass of results.* Only about half of AM's forays were dead-ends, and most of those looked promising initially.

4. *By the character of the human-machine interactions.* AM was never pushed far along this dimension. The human "user" is really little more than an observer, a monitor; he occasionally interrupts AM to ask one of a few possible questions, or to rename some common concept, etc.

5. *By its informal reasoning abilities.* AM was able quickly to "guess" the truth value of its conjectures, to estimate the overall worth of each new concept, to zero in on plausible things to do each cycle, and to notice glaring analogies (sometimes).

6. *By the results of experiments – and the fact that experiments could be performed at all on AM.* See Sec. 3.3.

7. *By future implications of this project.* Only time will tell whether this kind of work will impact on how mathematics is taught (for example, explicit teaching of heuristics?), on how empirical research is carried out by scientists, on our understanding of such phenomena as discovery, learning, and creativity, etc.

8. *By comparisons with other, similar systems.* Some of the techniques AM uses were pioneered earlier: for example, prototypical models (Gelernter 1963), and analogy (Evans 1968 and Kling 1971). There have been many attempts to incorporate heuristic knowledge into a theorem prover (Wang 1960, Guard 1969, Bledsoe 1971, Brotz 1974, Boyer and Moore 1975). Most of the apparent differences between them and AM vanish upon close examination: The goal-driven control structure of these systems is a compiled form of AM's rudimentary "focus of attention" mechanism. The fact that their overall activity is typically labelled as deductive is a misnomer (since constructing a difficult proof is usually in practice quite inductive). Even the character of the inference processes are analogous: The provers typically contain a couple of binary inference rules, like Modus Ponens, which are relatively risky to apply but can yield big results: AM's few "binary" operators have the same characteristics: Compose, Canonize, Logically-combine (disjoin and conjoin). The deep distinctions between AM and the "heuristic theorem provers" are these: the underlying motivations (heuristic modelling *vs* building tools for problem solving), the richness of the knowledge base (hundreds of heuristics *vs* only a few), and the amount of emphasis on formal methods.

Theory formation systems in any field have been few. Meta-Dendral (Buchanan 1975) represents perhaps the best of these. But even that program is given a fixed set of templates for the bond-breaking rules which it wishes to find, and a fixed vocabulary of mass spectral concepts to plug into those hypothesis templates; whereas AM selectively enlarges its vocabulary of mathematical concepts.†

There has been very little published thought about "discovery" from an algorithmic point of view; even clear thinkers like Polya (1954) and Poincaré (1929) treat mathematical ability as a sacred, almost mystic quality, tied to the unconscious. The writings of philosophers and psychologists invariably attempt to examine human performance and belief, which are far more manageable than creativity *in vitro.*†

Amarel (1967) notes that it may be possible to learn from "theorem finding" programs how to tackle the general task of automating scientific research. AM has been one of the first attempts to construct such a program.

† Also note that unlike Meta-Dendral, AM must gather its own data. On the other hand, this is mich easier in mathematics than in organic chemistry.

† It is not clear how to design a null hypothesis experiment, one with a control group, for tasks in which real scientists are performing real research. Hence psychologists simply don't study such human activities. This is the danger Kuhn (1970) warns us against, of becoming "paradigm-locked".

3.5 Final conclusions

- AM is a demonstration that a few hundred general heuristic rules suffice to guide an automated mathematics researcher as it explores and expands a large but incomplete knowledge base of mathematical concepts. Results indicate that some aspects of creative research can be effectively modelled as heuristic search.
- This work has also introduced a control structure based upon an ordered agenda of small research tasks, each with a list of supporting reasons attached.
- The main limitations of AM was its inability to synthesize powerful new heuristics for the new concepts it defined.
- The main successes were the few novel ideas it came up with, the ease with which a new task domain was fed to the system, and the overall rational sequences of behaviour AM exhibited.

Acknowledgements

This research was initiated as my PhD thesis at Stanford University (Lenat 1976), and I wish to deeply thank my advisers and committee members: Bruce Buchanan, Paul Cohen, Edward Feigenbaum, Cordell Green, Donald Knuth, and Allen Newell. In addition, I gladly acknowledge the ideas received in discussions with Avra Cohn and with Herbert Simon.

This work was supported in part by the National Science Foundation (MCS77-0440) and in part by the Defense Advanced Research Projects Agency (F44620-73-C-0074).

REFERENCES

Amarel, S. (1967). On representations and modelling in problem solving and on future directions for intelligent systems. *RCA Labs Scientific Report No. 2,* Princeton: Princeton University.

Bledsoe, W. W. (1971). Splitting and reduction heuristics in automatic theorem proving. *Artificial Intelligence* **2**, 55–77.

Boyer, R. S and Moore, J. S. (1975). Proving theorems about LISP functions. *J. ACM* **22**, No. 1, 129–144.

Brotz, D. K. (1974). Embedding heuristic problem solving methods in a mechanical theorem-prover. *STAN-CS-74-443.* Stanford: Dept. of Computer Science, Stanford University.

Buchanan, B. G. (1975). Applications of Artificial Intelligence to Scientific Reasoning. *Proc. USA–Japan Computer Conference (AFIPS and IPSJ),* Tokyo, pp. 189–194. New Jersey: American Federation of Information Processing Societies.

Buchanan, B. G., Lederberg, J. and McCarthy, J. (1976). Three Reviews of J. Weizenbaum's "Computer Power and Human Reason". *SAIL AIM-291,* Stanford: Artificial Intelligence Laboratory, Stanford University.

Davis, R. (1976). Applications of meta-level knowledge to the construction, maintenance and use of large knowledge bases. *SAIL AIM-291,* Stanford: Artificial Intelligence Laboratory, Stanford University.

Evans, T. G. (1968). Program for the solution of geometric-analogy intelligence test questions, *Semantic Information Processing,* pp. 271–353. (ed. Minsky, M. L.). Cambridge, Mass.: MIT Press.

Feigenbaum, E. A., Buchanan, B. G. and Lederberg, J. (1971). On generality and problem-solving: a case study using the DENDRAL program. *Machine Intelligence 6,* pp. 165–190, (eds Meltzer, B. and Michie, D.). Edinburgh: Edinburgh University Press.

Gelernter, H. (1963). Realization of a geometry-theorem proving machine. *Computers and Thought*, pp. 134–152, (eds Feigenbaum, E. A. and Feldman, J.) New York: McGraw Hill.

Guard, R., et al (1969). Semi-automated mathematics. *J. ACM* 16, 49–62.

Kling, R. E. (1971). Reasoning by analogy with applications to heuristic problem-solving: a case study. *AI Memo AIM-147*, Stanford: Artificial Intelligence Laboratory, Stanford University.

Kuhn, T. S. (1970). *The Structure of Scientific Revolutions*, 2nd ed. Chicago: Chicago University Press.

Lenat, D. B. (1976). AM; an artificial intelligence approach to discovery in mathematics as heuristic search. *SAIL AIM-286.* Stanford: Artificial Intelligence Laboratory, Stanford University.

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

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

Poincaré, H. (1929). *The Foundations of Science: science and hypothesis; the value of science; science and method.* New York: The Science Press.

Polya, G. (1954). *Mathematics and Plausible Reasoning.* Princeton: Princeton University Press.

Shortliffe, E. H. (1974). MYCIN – a rule-based computer program for advising physicians regarding antimicrobial therapy selection. *SAIL AIM-251*: Stanford: Artificial Intelligence Laboratory, Stanford University.

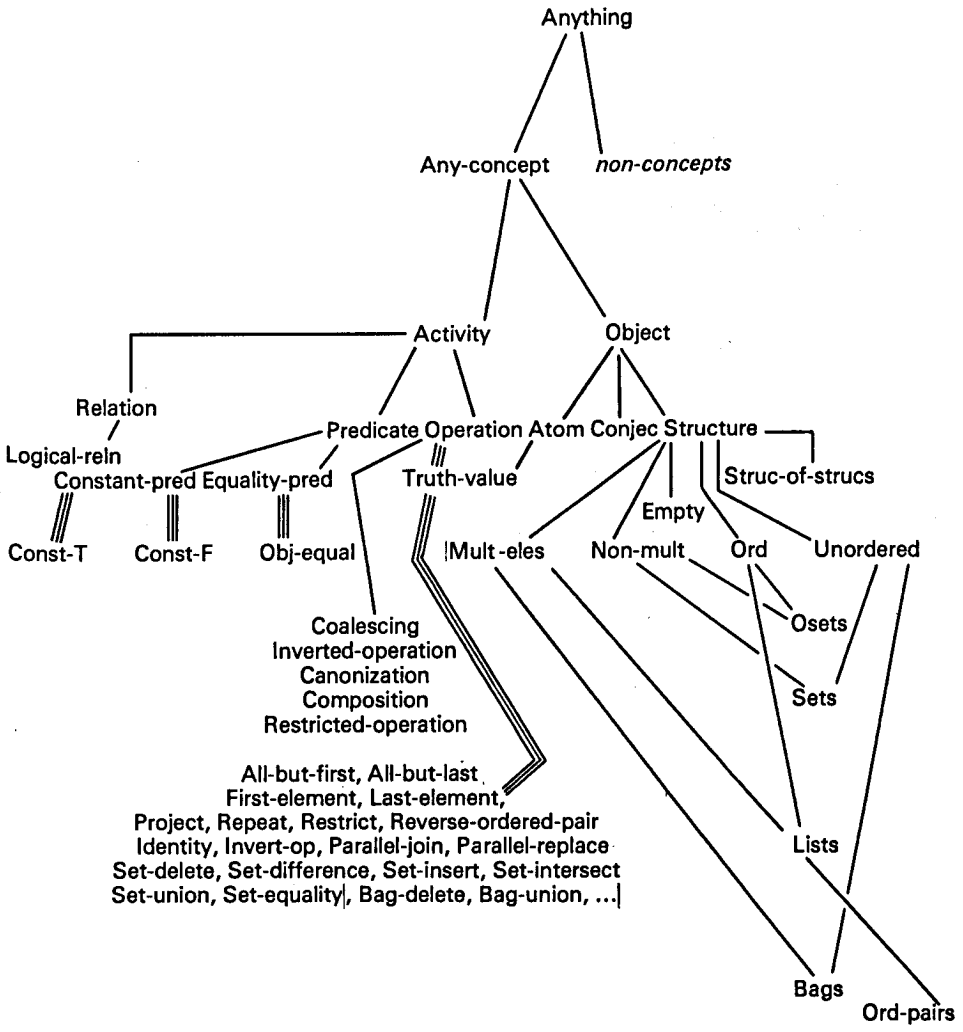
Wang, H. (1960). Towards mechanical mathematics. *IBM Journal of Research and Development* 4, *No. 1*, 2–22.

Winston, P. H. (1970). Learning structural descriptions from examples. *TR-231.* Cambridge, Mass.: Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

APPENDIX I

Concepts initially given to AM

Below is a graph of the concepts which were present in AM at the beginning of its run. Single lines denote Generalization/Specialization links, and triple lines denote Examples/Isa links.



APPENDIX II

Concepts discovered by AM

The list below is meant to suggest the range of AM's creations; it is far from complete, and many of the omissions were real losers. The concepts are listed in order in which they were defined. In place of the (usually awkward) name chosen by AM, I have given either the standard mathematics/English name for the concept, or else a short description of what it is.

Sets with less than 2 elements (singletons and empty sets).
Sets with no atomic elements (nests of braces).
Bags containing (any number of copies of) just one kind of element.
Superset (contains).
Doubleton bags and sets.
Set-membership.
Disjoint bags.
Subset.
Disjoint sets.
Same-length.
Same first element.
Count (Length).
Numbers (in unary).
Add.
Minimum.
SUB1 (λ (x) x-1).
Subtract (except: if $x<y$, then x-y results in zero).
Less than or equal to.
Times.
Compose a given operation F with itself (form F-o-F).
Insert structure S into itself.
Try to delete structure S from itself (a loser).
Double (add 'x' to itself).
Subtract 'x' from itself (as an operation, this is a real zero).
Square (λ (x) Times(x,x)).
Coalesced-join: (λ (S F) append together F(s,s), for each $s \in S$).
Coalesced-replace: replace each element s of S by F(s,s).
Coa-repeat2: create a new op which takes a struc S, op F,
and repeats F(s,t,S) all along S.
Compose three operations: λ(F,G,H) F-o-(G-o-H).
Compose three operations: λ(F,G,H) (F-o-G)-o-H.
Add^{-1} (x): all ways to repr. x as the sum of nonzero nos.
G-o-H, s.t. H(G(H(x))) is always defined (wherever H is).
Insert-o-Delete; Delete-o-Insert.
Size-o-Add^{-1}. (λ (n) The number of ways to partition n).
Cubing.
Exponentiation.
Halving (in natural numbers only; thus Halving(15)=7).
Even numbers.
Integer square-root.
Perfect squares.
Divisors-of.
Numbers-with-0-divisors; Numbers-with-1-divisor.
Primes (Numbers-with-2-divisors).
Squares of primes (Numbers-with-3-divisors).
Squares of squares of primes.
Square-roots of primes (a loser).
$Times^{-1}$ (x): all ways of repr. x as the product of nos. (>1).

All ways of representing x as the product of primes.
All ways of representing x as the sum of primes.
All ways of representing x as the sum of two primes.
Numbers uniquely representable as the sum of two primes.
Products of squares.
Multiplication by 1; by 0; by 2.
Addition of 1; of 0; of 2.
Product of even numbers.
Sum of squares.
Sum of even numbers.
Pairs of squares whose sum is also a square ($x^2+y^2=z^2$).
Prime pairs ($\{(p,q,r) \mid p,q,r$ are primes $\wedge\ p+q=r\}$).

APPENDIX III

A brief, task-by-task trace

1. Fill in examples of Compose. Failed, but suggested next task:
2. Fill in examples of Set-union. Also failed, but suggested:
3. Fill in examples of Sets. Many found (e.g., by instantiating Set.Defn) and then more derived from those examples (e.g., by running Union.Alg).
4. Fill in specializations of Sets (because it was very easy to find examples of Sets). Creation of new concepts. One, INT-Sets, is related to "Singletons". Another, "BI-Sets", is all nests of braces (no atomic elements).
5. Fill in examples of INT-Sets. This indirectly led to a rise in the worth of Equal.
6. Check all examples of INT-Sets. All were confirmed. AM defines the set of Nonempty INT-Sets; this is renamed "Singletons" by the user.
7. Fill in examples of Bags.
8. Check examples of Bags. Defined INT-Bags and BI-Bags.
9. Fill in examples of All-but-first.
10. Fill in examples of All-but-last.
11. Fill in specializations of All-but-last. Failed.
12. Fill in examples of List-union.
13. Fill in examples of Projl.
14. Check examples of All-but-first.
15. Fill in examples of Empty-structures. 4 found.
16. Fill in generalizations of Empty-structures. Failed.
17. Fill in examples of Set-union.
18. Check examples of Set-union. Define λ (x,y) $x \cup y = x$, later called Superset.
19. Fill in examples of Bag-insert.
20. Check examples of Bag-insert. Range is really Nonempty bags. Isolate the results of insertion restricted to Singletons: call them Doubleton-bags.
21. Fill in examples of Bag-intersect.
22. Fill in examples of Set-insert.
23. Check examples of Set-insert. Range is always Nonempty sets. Define λ (x,S) Set-insert(x,S)=S; i.e., set membership. Define Doubleton sets.
24. Fill in examples of Bag-delete.
25. Fill in examples of Bag-difference.
26. Check examples of Bag-intersect. Define λ (x,y) $x \cap y = ()$; i.e. disjoint bags.
27. Fill in examples of Set-intersect.
28. Check examples of Set-intersect. Define λ (x,y) $x \cap y = x$; i.e., subset. Also define disjoint sets: λ (x,y) $x \cap y = \{\}$.
29. Fill in examples of Equal. Very difficult to find examples; this led to:
30. Fill in generalizations of Equal. Define "Same-size", "Equal-CARs", and some losers.
31. Fill in examples of Same-size.

32. Apply an Algorithm for Canonize to the args Same-size and Equal. AM eventually synthesizes the canonizing function "Size". AM defines the set of canonical structures: bags of T's; this later gets renamed as "Numbers".
33. Restrict the domain/range of Bag-union. A new operation is defined, Number-union, with domain/range entry ⟨Number Number→Bag⟩.
34. Fill in examples of Number-union. Many found.
35. Check the domain/range of Number-union. Range is 'Number'. This operation is renamed "Add2"; it adds any two natural numbers.
36. Restrict the domain/range of Bag-intersect to Numbers. Renamed "Minimum".
37. Restrict the domain/range of Bag-delete to Numbers. Renamed "SUB1".
38. Restrict the domain/range of Bag-insert to Numbers. AM calls the new operation "Number-insert". Its domain/range entry is ⟨Anything Number→Bag⟩.
39. Check the domain/range of Number-insert. This doesn't lead anywhere.
40. Restrict the domain/range of Bag-difference to Numbers. This becomes "Subtract".
41. Fill in examples of Subtract. This leads to defining the relation LEQ (≤).
42. Fill in examples of LEQ. Many found.
43. Check examples of LEQ.
44. Apply algorithm of Coalesce to LEQ. Conjecture: LEQ(x,x) is Constant-True.
45. Fill in examples of Parallel-join2. Included is Parallel-join2(Bags,Bags'Proj2), which is renamed "TIMES", and Parallel-join2(Structures,Structures,-Prolj1), a generalized Union operation renamed "G-Union", and a bunch of losers.
46. Fill in and check examples of the operations just created (really several tasks).
47. Fill in examples of Coalesce. Created: Self-Compose, Self-Insert, Self-Delete, Self-Add, Self-Times, Self-Union, etc. Also: Coa-repeat2, Coa-join2, etc.
48. Fill in examples of Self-Delete. Many found.
49. Check examples of Self-Delete. Self-Delete is just Identity-op.
50. Fill in examples of Self-Member. No positive examples found.
51. Check examples of Self-Member. Self-Member is just Constant-False.
52. Fill in examples of Self-Add. Many found. User renames this "Doubling".
53. Check examples of Coalesce. Confirmed.
54. Check examples of Add2. Confirmed.
55. Fill in examples of Self-Times. Many found. Renamed "Squaring" by the user.
56. Fill in examples of Self-Compose. Defined Squaring-∘-Squaring. Created Add-∘-Add (two versions: Add21 which is λ (x,y,z) (x+y)+z, and Add22 which is x+(y+z)). Similarly, two versions of Times-∘-Times and of Compose-∘-Compose.
57. Fill in examples of Add21. (x+y)+z. Many are found.
58. Fill in examples of Add22. x+(y+z). Again many are found.
59. Check examples of Squaring. Confirmed.
60. Check examples of Add22. Add21 and Add22 appear equivalent. But first:

61. Check examples of Add21. Add21 and Add22 still appear equivalent. Merge them. So the proper argument for a generalized "Add" operation is a Bag.
62. Apply algorithm for Invert to argument 'Add'. Define Inv-add(x) as the set of all bags of numbers (>0) whose sum is x. Also denoted Add-1-(x).
63. Fill in examples of TIMES21. (xy)z. Many are found.
64. Fill in examples of TIMES22. x(yz). Again many are found.
65. Check examples of TIMES22. TIMES21 and TIMES22 may be equivalent.
66. Check examples of TIMES21. TIMES21 and TIMES22 still appear equivalent. Merge them. So the proper argument for a generalized "TIMES" operation is a Bag. Set up an analogy between TIMES and ADD, because of this fact.
67. Apply algorithm for Invert to argument 'TIMES'. Define Inv-TIMES(x) as the set of all bags of numbers (>1) whose product is x. Analogic to Inv-Add.
68. Fill in examples of Parallel-replace2. Included are Parallel-replace2(Bags, Bags,Proj2) (called MR2-BBP2), and many losers.
69. Fill in an check examples of the operations just created.
70. Fill in examples of Compose. So easy that AM creates Int-Compose.
71. Fill in examples of Int-Compose. The two chosen operations G,H must be such that ran(H)*dom(G), and ran(G)*dom(H); both G and H must be interesting. Create G-Union-o-MR2-BBP2, Insert-o-Delete, Times-o-Squaring, etc.
72. Fill in and check examples of the compositions just created. Notice that G-Union-o-MR2-BBP2 is just TIMES.
73. Fill in examples of Coa-repeat2. Among them: Coa-repeat2(Bags-of-Numbers, Add2) [multiplication again!], Coa-repeat2(Bags-of-Numbers, Times) [exponentiation], Coa-repeat2(Structures, Proj1) [CAR], Coa-repeat2-(Structures, Proj2) [Last-element-of], etc.
74. Check the examples of Coa-repeat2. All confirmed.
75. Apply algorithms for Invert to 'Doubling'. The result is called "Halving" by the user. AM then defines "Evens", and also "Odds".
76. Fill in examples of Self-Insert.
77. Check examples of Self-Insert. Nothing special found.
78. Fill in examples of Coa-repeat2-Add2.
79. Check examples of Coa-repeat2-Add2. It's the same as TIMES.
80. Apply algorithm for Invert to argument 'Squaring'. Define "Square-root".
81. Fill in examples of Square-root. Some found, but very inefficiently.
82. Fill in new algorithms for Square-root. Had to ask user for a good one.
83. Check examples of Square-root. Define the set of numbers "Perfect-squares".
84. Fill in examples of Coa-repeat2-Times. This is exponentiation.
85. Check examples of Coa-repeat2-Times. Nothing special noticed, unfortunately.
86. Fill in examples of Inv-TIMES. Many found, but inefficiently.
87. Fill in new algorithms for Inv-TIMES. Obtained opaquely from the user.
88. Check examples of Inv-TIMES. This task suggests the next one:

89. Compose G-Union with Inv-TIMES. Good domain/range. Renamed "Divisors-of".
90. Fill in examples of Perfect-squares. Many found.
91. Fill in examples of Divisors-of.

This is where the excerpt presented in Sec. 3.1 begins: primes are defined, and AM soon discovers some unexpected (and hence, interesting) relationships involving them, which makes the Worth rating of Primes increase greatly.