AUTOMATED THEORY FORMATION IN MATHEMATICS[1]

Douglas B. Lenat
Computer Science Department
Carnegie-Mellon University
Pittsburgh, Pa. 15213

Abstract

A program called "AM" is described which cairies on simple mathematics research: defining, and studying new concepts under the guidance of a large body of heuiistic rules. The 250 heurKtus communicate via an agenda mechanism, a global priority queue of small bisk', for the program to pei foim and teasons why each task is plausible (e.g., "Find PENCRAHZTION. of 'prnes', because turued out to be so useful a Conccpi"). Fach concept is an active, structured knowledge module. One bundled vei y incomplete modules are initially supplied, each one corresponding to an elementary set theoretic concept (e.g., union). This provides a definite but immense space which AM begins to explore. In one boor, AM rediscovers hundreds of common concepts (including singleton sets, natural numbers, arithmetic) and theorems (e.g., unique factorization).

1. INTRODUCTION

1.1. HISTORICAL MOTIVATION

Scientists often face the difficult task of formulating nontrivial research problems which are soluble. In most brandies of science, it is usually easier to tackle a specific given problem than to propose interesting yet managable 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 versus *proposing* it in the first place.

A wealth of AI research has been focussed upon the former type of activity: deductive problem solving (see, e.g., [Bledsoe 71], [Nilsson 71], [Newell & Simon 72]). Approaches to *inductive* inference have also been made. Some researchers have tried to attack the problem in a completely domain-independent way (see, e.g., [Winston 70]). Other AI researchers believe that "expert knowledge" must be present if inductive reasoning is to be done at the level which humans are capable of. 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 74], mass spectra identification [Feigenbaum 71], clinical dialogue [Davis 76], discovery of new mass spectroscopy rules [Buchanan 75].

The "next step" in this progression of tasks would be that of fully automatic theory formation in some scientific field. "1 his includes two activities: (i) discovering relationships among known concepts (e.g., by formal manipulations, or by noticing regularities in empirical data), and (ii) defining new concepts for investigation. Meta-Dendral [Buchanan 75] performs only the first of these; most domain-independent concept learning programs (Winston *70]* 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.

What we are describing is a computet 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, concentrate upon the most interesting ones, *and* 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, beeausc?:

1. there are no uncertainties in the raw data (e.g., arising from errorful measuring devices).
2. Reliance on exports' 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 (e.g., the less one needs to use natural language to communicate information).
4. A mathematician has the freedom to explore -- or to give up on -- whatever he wants to. There is no specific problem to solve, no fixed "goal".
5. Unlike some fields (e.g., propositional logic), elementary math research has an abundance (many hundreds) of powerful heuristic rules available.

The limitations of math 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" behavior. Since math has been worked on for millcnia 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.

1.3 INITIAL ASSUMPTIONS AND HYPOTHESES

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

1. Very little natural language processing capabilities are

Thie work was supported in part by the Defense Advanced Research Projects Agency (M1620-73-C-0074) and monitored by the Air rorce Office of Scientific Research

required. As it runs, AM is monitored by a human *"user".* AM Keeps the user informed by instantiating English sentence templates. 1 he 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). Tor each new piece of Knowledge gained, there will be no trouble in finding which facet of which concept it should he 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 tasKs. The techniques for actually carrying out a tasK are contained within a large collection of heuristics.

b. Each heuristic has a well defined domain of applicability, which coincides perfectly with one of AM's concepts. We say the heuristic "belongs to" that concept

6. Heuristics superimpose; they never interact strongly with each other. If one concept CI is a specialization of concept C?, then CI's heuristics are more powerful and should he tried first.

7. The reasons supporting a tasK (on the agenda of facet/concept tasKs to be carried out) superimpose peifectly. "they never change with time, and it maKes no difference in what order they were noticed. It suffices to have a single, positive number which characterizes tire value of the reason.

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.

9. The set of heuristics need not grow, as new concepts are discovered. All common-sense 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 doable (by one person, in one year).

One point of agreeement between Wei/enbaum and lederbeig [lederberg 76] is that AI can succeed in automating only those activities for which there exists a "strong theory" of how that activity is done by people. Point #4 above is a claim that such a clean, simple model exists for math research: a search process governed by a large collection of houristic rules. Here is a simplified summary of that model:

1. the order in which a math textbook presents a theory is almost the exact opposite of the order in wheh it war. actually developed. In a text, definitions and lemmata 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 begin by examining some already-Known concepts, trying to find some regularity involving them, formulating those as conjectures to investigate further, and using them to motivate some simplifying new definitions.
2. Each step the researcher taKes (see #1) 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 interestingness, empirical induction, analogy, utility estimates) are much more important than formal methods.
4. All such heuristics can be viewed as situation/action (IF/IHLN) rules. There is a common core of (a few hundred) heuristics, basic to all fields of math at all levels. In addition to these, each field has several specific, powerful rules.
5. Nature is metaphysically pleasant: It is fair, uniform, regular. Statistical considerations are valid and valuable when trying to find regularity in math data. Simplicity and synergy and symmetry abound.

2. DESIGN OF THE 'AM' PROGRAM

A pure production system may be considered to consist of thtee 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 math 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 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 *2b* 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 whic h it answers.

Name: What shall wo call C when talking with the user?
Generalizations: Which other concepts have less restrictive (i.e., weaker) definitions than C?
Specialisations: Which concepts satisfy C's definition plus some additional constraints?
Examples; What 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 if we want to execute the operation C on a given argument?
Domain/Range: 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.)
Interestingness: What special features can make a C especially interesting? Especially boring[9]

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 filling in new entries for facet F of a concept which is a C?
F.CheckK: How do we verify/debug potential entries?
F.Suggest: If AM bogs clown, what are some new tasKs (related to facet F of concept C) to consider doing?

In the Lisp implementation of AM, each (oncept 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. As an example, here is *a* rendition of the Sets concept. It is meant to correspond to the notion of a collection of elements.

```
Name(s): Set, Class, Collection
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
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}
    Fable: <4,1,A,>
Conjecs: 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-prod, 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 notation C.F will refer to applying the corresponding function. Go the first entry in the Definitions facet is recursive because it contains an embedded call on the function Set.Definition. Since there are three separate but equivalent definitions, AM may choose whichever one it wants when it recurs. AM can choose 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 two specific concepts (e.g., 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 to a sef of concepts (e.g., between {bags, hag-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 (e.g., 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 can not pick up new intuitions via metaphors to the real world, as we can).

Since "Sets" is a static concept, it had no Algorithms facet (as did, e.g., "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 or Transparent? Destructive?
2. Relators: Is, this just a special case of some other concept's algorithm? Which others does this one call on[7] is this similar to any other algorithms?
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 arc multiple algorithms because different ones have different properties: some are very quick in some cases, some are always slow but are very cleanly written and hence 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 •D1 D2... Di --> R>, which means that the concept takes a list of arguments, the first one being an example of concept D1, the second of D2,..., the last argument being an example of concept Di, and if the algouthm (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 D1 xD2x...xDi, and that the Range of the concept is R. for example, the Domain/iange 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.

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 wore selected, corresponding roughly to what Piaget might have called "prenumerical knowledge". Figure J presents a graph of these concepts, showing their interrelationships of Genoralization/Specialiation and Fxamples/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 get defined, this number increases. IPs worth having AM spend some time deciding which basic task to work on next, for two reasons: most of the tasks will *never* get 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. Fach 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 constrainings 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 (ontrol policy is to plucK the top task (highest priority rating) from the agenda, and then execute it. While a task executes, some new bisks may he proposed *(and* merged into the agenda), some new concept', may get created, and (hopefully) some entries for the specified facet of the specified concept will he found and filled in. Once a task ir. chosen, the priority rating of that task now serves a new function: it is taken as an estimate of how much computational resource to devote to working on this task. 1 he task above, in the box, might he 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. Eg.,

IF the cunent task is to check examples of a concept X,
and (Forsome Y) Y is a generahzation of X,
and Y has al least 10 known examples
and all examples of Y are also examples of X,
THKN conjecture: X is really no more speciali/.ed than Y,
and add that conjecture as a new entry on the
Kxamples facet of the ConJOCK 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 genetalizations.

It is the heuristics" right hand sides which actually accomplish the selected task; that process will be described in the next subsection. The left sides are the relevancy checkers, and will be focussed on now:

Syntactically, the left side must be a predicate, a Lisp function which always returns True or False. It must be a conjunction P1AP2AP3A„. of smaller predicates Pi, each of which must be quick and must have no side effects. Here are some typical conjucts which might appear inside a left hand side:

Over half of the current task's tunc allotment, is used up;
There are some known examples of Structures;
Some known generalization of the current concept (the concept mentioned as part o| the current task) has a completely empty Kxamplcs 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 he fore;

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 analyze and modify its own heuristics.

f rom 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 repeatedly evaluating each one's condition would use up tremendous amounts of time. AM is able to quickly 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 riitTr.nl ta^k * to find examples of activity F,
and a fast algorithm for computing F is known,
THEN one.way to get examples of F is to run F on landomly chosen examples of the Domain of F.

The very first conjunct of a rule's left side is always special. It specifics 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-precondilions: A *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 "(duck the Domain/range facet of concept X", where X is some surjectivc 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.

When a task is chosen, it specifies a concept C and a facet F 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* generalisations, etc. If the current task were "Check the Domain/range of 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 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 deal with Domain/range facets of any concept, etc. Of course, the

This operation is the result of composing set-union with itself. It performs X (x,y,z) xu(yuz).

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 ninnut task is "Check the Domain/range of F",
and an entry on that facet has the form <D D...1) -> H>,
and concept R is a generalization of rnncrpl I),
THKN it is worth spending time checking whether or not
the range of F rnight he simply I), instcad of R.

Suppose 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,

Merc is another way to look at the heuristic-gathering process. All the concepts known to AM are arranged in a big hierarchy, via subsetof links (Specializations) and element-of links (Isa). Since each heuristic is associated with one individual concept (its domain of applicability), there is a hierarchy induced upon the set of heuristics. Heritability properties hold: a heuristic tacked onto concept C is applicable to working on all "lower" concepts. This allows us to efficiently analogically access the 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. Fach 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.
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. Once the potentially relevant rules are located (in step 2), AM evaluates eaclVs 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. 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 the value it returns is always 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 situations 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.

Dear 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:

73.7A 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, to numerically 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:

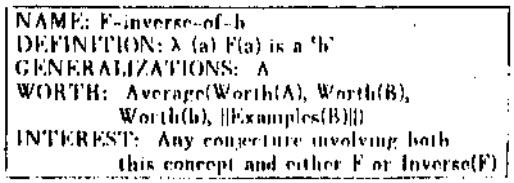
IF the current task was "Fill in examples of X",
and concept X is a Predicate,
and over 100 items are known in the domain of X,
and at least 10 cpu sees, have been spent so far,
and X has returned True at least once,
and X returned False over 20 times as often as True,
THKN add the following task to the agenda:
"Fill in gcneraltations of X"
for the following reason:
"X is rarely satisfied; a slightly less restrictive
concept might he much more ml cresting"
This reason has a rating which is the False/True 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 to 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".

73.7,2 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 exist on the agenda, tasks which indicate that AM ought to spend some time filling in some of those facets in the near future. Here is a heuristic rule which results in a new concept being created:

IF the current task was "Fill in examples of F"
and F is an operation, 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 distin-
guished member (especially, an extremum) of B,
THKN for each such 'b'B, create the following concept:



and the reason for this neat ion is: "It's worth investigating A's whieh havr unusual F-values"
and add five new tasks to the anemia,
of the form "Kill in faeet x of F-inveisr-oi-h"
where x is Coinertures, Generlions, Specialization, Kxamples, and Isa's;
for the following reason:
"This conecpt was newly synthesized; it is erurial lo find where it 'fits in' to the hierarehy"
Thr reason's rating is just Worlh(Finverse-of-b).

One use of this heuristic was when thr current task was "F ill in examples of Divisors~of". The heuristics 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 (e.g., singletons, empty sels, 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:

NAMK: Divisors-ol - InvrrseoK Ooubleton
OKFINITION: X (a) Divisors of(a) is a Don Melon
GENKKAU/ATI0NS: Numbers
WORTH: 100
INTKRKST: Any conjecture involving both this eoneept and eilher 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 'Till 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.

There is just a tiny bit of "theory" behind how these concept-creating rules were designed. A facet of a new concept is filled in immediately 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. Each other facet is either left unmentionod 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 many new tasks at the moment that Primes were created, including "Fill in conjectures about Primes", "Fill in specializations of Primes", etc.

23.23 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 applicable to any Activity:

IF the curreut task is to fill in examples of F,
and F is an operation, say with domain I),
and there is a fast known algorithm for F,
THKN one way to get examples of F is lo run F'S algorithm on randomly chosen examples of I).

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 malehes (FILLIN EXAMPLES F*-anything),
and F isa Aetivily,
and l.he Algorithms farel of F is not blank,
THKN carry out the following proredure:
1. Find the domain of F, and rail it D;
2. Find examples of D, and rail them K;
3. Find a fast algorithm to compute F; call it A;
4. Repeatedly:
4a. Choose any member of E, and call it KI.
4b. Run A on E1, and call the result X.
4e. Check whether <E1,X> satisfies the definition of F.
4d. If so, then add 'I'll -> X> to the Kxarnples facet of F.
4e. If not, then add <K1 -> 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 SFT -> SLT), 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 *(?)* 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 {7}, {A,B,C,D,F), {}, {A,{fB}},...

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 (4a) has us choose one pair of our examples of sets, say the first two (7} and (A,B,C,D,E). Step (4b) has us run UNION on these two sets. The result is {A,B,C,D,F,7}. Step (4c) has us grab an entry from the Definitions facet of Set-union, and run it. A typical definition is this formal one:

```
(X (S1 S2 S3)
      (AND
            (For all x in SI, x is in S3)
            (For all x in S2, x is in S3)
            (For all x in S3, x is in S1 or x is in S2))))
```

It is run on the three arguments S1={Z}, S2={A,B,C,D,E}, S3~{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. The loop ends when either the time or space allotted to this

rule is exhausted. AM would then 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. Here is a tiny excerpt:

```
+* Task: **   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 v. int.
        (3) Focus of attention: AM just defined Divisors-of.
26 examples found, in 9 secs, e.g., Divisors- of(6)={ 1,2,3,6}.

** Taski **  Consider nos. having small sets of Divisors-of.
    2 Reasons:
        (1) Worthwhile to look for extreme cases.
        (2) Focus: AM just worked on Divisors-of.
Filing  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.
  J examples found, in 4 sees, e.g., Divisors of(I) = {1J.
  Conjecture: 1 is the only number with exactly 1 divisor.
Filling in examples of numbers with 2 divisors.
  24 examples found, in 4 sees. 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.
  1 1 examples found, in 4 secs. Divisors-of(49)={ 1,7,49}.
  All nos. with 3 divisors are also Squares. Unexpected!.
  Creating a new concept: "Numbers-with-3-divisors".

** Task: **  Consider square-roots of Nos-with-3-divisors.
    2 Reasons:
        (1) Numbers-with-3 divisors unexpectedly turned
              out to all be Perfect Squares as well.
        (?) Focus: AM just defined Nos-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.
      The chance of coincidence is below acceptable limits.
  Boosting the Worth rating of both concepts.

** TasK:** Consider the squares of Nos-witlv-3-divisors.
    3 Reasons:
        (1) Squares of Nos-with-2~divisors were v. int.
        (2) Square-roots of Nos-with-3-divisors were int.
        (3) Focus: AM just worked on Nos-with-3-divisors.
```

3.2. OVERALL PERFORMANCE

Now that we've seen how AM works, and we've been exposed to a bit of "local" results, let's take a moment to discuss the totality of the mathematics which AM carried out. AM began its investigations with scanty knowledge of a hundred elementary concepts of finite set theory (see Fig. 1). 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 in divisibility theory. See Fig. 2. Prime pairs, Diophantine equations, the unique factorization of numbers into primes, Goldbach's. conjecture -- these were some of the *nice* discoveries by AM. Many concepts which we know to be crucial were never uncovered, however: remainder, gcd, greater-than, infinity, proof, etc. These "omissions", *could* have been discovered by the existing heuristic rules in AM. 1 he paths which would have resulted in their definition were simply never rated high enough to explore.

All the discoveries mentioned (including those in Fig. 2) were made in a run lasting one cpu hour (Interlisp + 100k, Sumex POP-10 KI). Two hundred jobs in toto were selected from the agenda and executed. On the average, a job was granted 30 cpu seconds, but actually used only 18 seconds. f or a typical job, about 35 rules were located as potentially relevant, and about a dozen actually fired. 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), and half the remaining Ones were only marginal.

Although AM fared well according to several different measures of performance (see Section 3.4), of great significance are its *Limitations.* As AM ran longer and longer, the concepts it defined were further and further from the primitives it began with. E.g., "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 *uniomng.* 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.

f or instance, one general rule indicated that AuB would be interesting if it possessed properties absent both from A and from 0. 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.

The key deficiency was the lack of adequate mera-rules ['Davis 76]: heuristics which reason about heuristics: keep track of their performance, modify them, create new ones, etc.

Aside from the preceding major limitation, most of the other problems pertain to missing knowledge: Many concepts one might consider basic to discovery in math are absent from AM; analogies were under-utilized; physical intuition was hand-crafted only; the interface to the user was far from ideal; etc. A large effort is underway this year at Carnegie-Mellon University, comprised of Greg Harris, Doug Lenat, Elaine Rich, Jim Saxe, and Herbert Simon, to overcome these limitations.

3.3. EXPERIMENTS_WJTHJAM'

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

3.3.1 Must the WORTH numbers be finely tuned?

Each of the 115 initial concepts had, supplied by the author, a rating number (0-1000) signifying its overall worth. The worth ratings affect the overall priority values of tasks on the agenda. Just how sensitive is AM*s behavior 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 wanderings (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 candidates 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 selected 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'?

Only 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 favor of having *symbolic justification* of the rank of each task in 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 [quality 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 (e.g., 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, Angle, Triangle, [quality of points/lines/angles/triangles). No new heuristics were added to AM.

AM was able to find examples of all the supplied concepts, *md to use the character of such empirical data to determine reasonable directions to proceed in its research. 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.

3.4. EVALUATING THE 'AM' PROGRAM

We may wish to evaluate AM 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.

?. 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.

'3, By the quality of the route AM took to accomplish this mass of results. Only about half of AMY 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.

5. By its informal reasoning abilities. AM was able to quickly "guess" the truth value of 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.

7. By future implications of this project. Only time will tell whether this kind of work will impact on how mathematics is taught (e.g., 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 to other, similar systems. Some of the techniques AM uses were pioneered earlier: e.g, prototypical models [Gelernter 63], and analogy [Evans 68], [Kling 71]. There have been many attempts to

incorporate heuristic knowledge? into a theorem prover [Wang 60], [Guard 69], [Bledsoe 71], [Brolz,-74], [Boyor & Moore *7b).* Most of the *apparent* differences, between them and AM vanish upon close examination: The goal-dtiven control structure of these systems is a compiled form of AM's; rudimentary "focus of attention" mechanism. "I he 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 binary inference rules, like Modus Ponens, which are relatively isky to apply but can yield hip results; AM's few "binary" operators have the same characteristics: Compose, Canonize, Logically-combine (disjoin and conjoin). The *deep* distinctions between AM and the "heurstic 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 *7b]* represents pethaps the best of these. Rut even this system is given a fixed set of templates for 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 math concepts. Also, AM must gather its own data, but this is much easier in math than in organic c hem.

There has been very little published thought about "discovery" from an algorithmic point of view; even clear thinkers like Polya and Poincate' 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 managable than creativity *in t3ro.* Amarel [J 967] notes 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.

3.5. FINAL CONCLUSIONS

-> AM is a demonstration that a few hundred general heuristic rules suffice to guide an automated math researcher as it explores and expands a large but incomplete knowledge base of math concepts. AM demonstrates 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 limitation 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 — most importantly — the overall rational sequences of behavior AM exhibited.

ACKNOWLEDGEMENT

This research was initiated as my Ph.D. thesis at Stanford University, 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 I have received in discussions with Avra Conn and with Herbert Simon.

REFERENCES

Amarel, Saul, *On Representations and Modelling in Problem Solving, and On Future Directions for Intelligent Systems,* RCA Labs Scientific Report No. 2, Princeton, N.J., 1967.

Bledsoe, W. W., *Splitting and Reduction Heuristics in Automatic Theorem Proving,* Artificial Intelligence 2, 1971, pp. 55-77.

Boyer , Robert S., and J S. Moore, *Proving Theorems about Lisp Pu.ncti.ons,* JACM, V. 22, No. 1, January, 1975, pp. 129 144.

Brotz, Douglas K., E*mbedding Heuristic Problem Solving Methods in a Mechanical Theorem Prover,* Stanford U. Report STAN CS74-44M, August, 1971.

Buchanan, Bruce G., *Applications of Artificial. Intelligence to Scientific Reasoning,* Second USA-Japan Computer Conference, published by AFIPS and IPS I, Tokyo, 1975, pp. 189-194.

Davis, Randall, *Applications of Meta Level Knowledge to the Construction, Maintenance, and Use of Large Knowledge Bases,* SAIL AIM-271, Artificial Intelligence Laboratory, Stanford University, July, 1976.

Evans Thomas G., *A Program for the Solution of Geometric- Analogy Intelligence Test Questions,* in (Minsky, Marvin, ed.), Semantic Information Processing, The MIT Press, Cambridge," Massachuisetts, 1968, pp. 2 71-353.

Heigenbaum, Edward, 0. Buchanan, and J. I.ederberg, *On Generality and Problem Sohung: A Cose Study Using 1 he DENDRAL Program,* in (Mellzcr and Michie, eds.) Machine Intelligence 6, .1971, pp 165 190.

Gclernter, K, *Realization of a Geometry-Thcorem Proving Machine,* in (Feigenbaum *and* Felclman, eds.) Computers and Ihoughtj McGraw Hill Book Company, New York, 1963, pp. 134-152.

Guard, James R., et al., *Semi Automated Mathematics,* JACM 16, January, 1969, pp. 49-62.

Klin};, Robert Elliot, *Reasoning by Analogy with Applications to Heurcstic Problem Solving: A Case Study,* Stanford AI Memo AIM-147, August, 1971.

Lederberg, Joshua, Review of J. Wcizenbaums's Computer Power and Human Reason, W. H. Freeman, ST., 1976.

Lenat, Douglas B., *AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search,* SAIL AIM-286, Artificial Intelligence Laboratory, Stanford University, July, 1976.

Newetl, Allen, and Herbert Simon, Human Problem Solving, Prentice-Hall, Englcwood Cliffs, New Jersey," 1972.'

Nilsson, Nils, Problem Solving Methods in Artificial LnteJliRonce, McGraw Hill", N.Y., 1971.

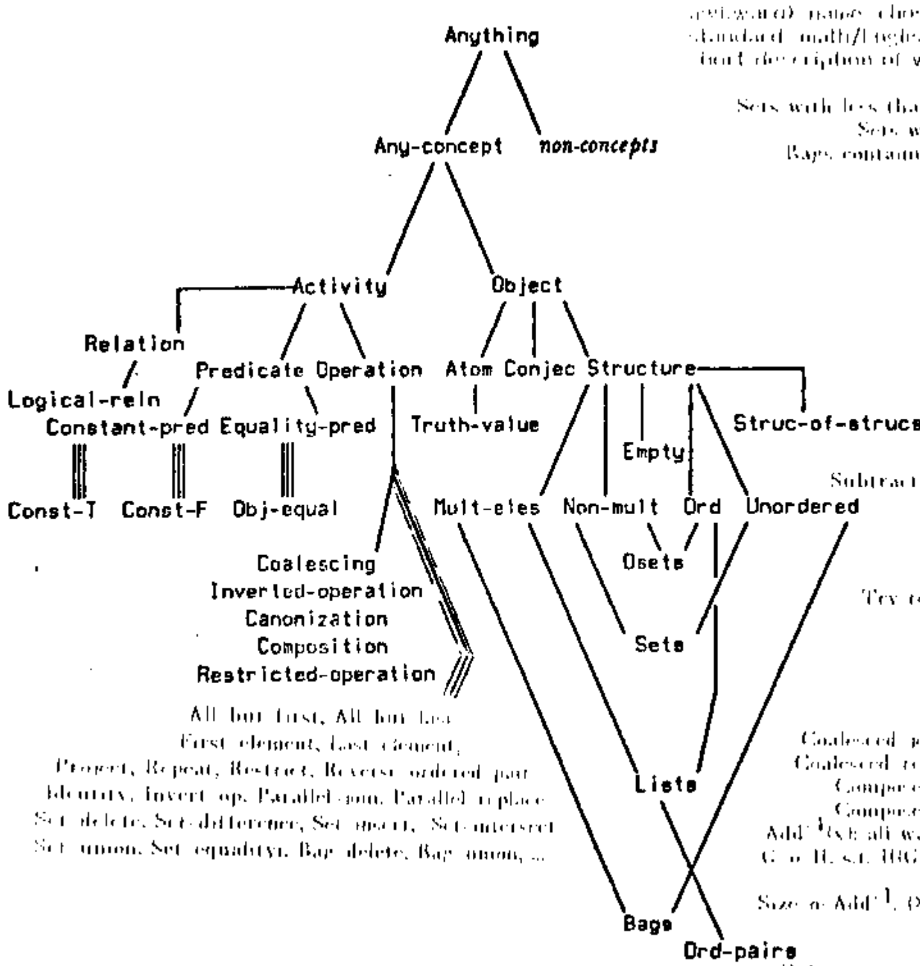
Shortliffc, E. H., *MYCIN — A rule-based computer program for advising physicians regarding antimicrobial therapy selection,* Stanford AI Memo 251, October, 1974.

Wanp„ Hao, *Toward Mechanical Mathematics,* IBM Journal of Research and Development, Volume 4, Number 1, January, 1960, pp. 2-22.

Winston, Patrick, *Learning Structural Descriptions from Examples,* TR-231, MIT AI Lab, September, 1970.

FIGURE 1: 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.



All but first, All but last
First element, Last element,
Project, Repeat, Restrict, Reverse ordered pair
Identity, Invert op, Parallel-join, Parallel replace
Set delete, Set-difference, Set insert, Set-intersect
Set union, Set equality, Bag delete, Bag union, ...

FIGURE 2: Concepis Discovcvci *by* AM

The list below is meant to suggest the range of AM's creations; it is far from complete, and many of the concepts were real losers. The concepts are listed in the order in which they were defined. In place of the (usually awkward) name chosen by AM, I have given either the standard math/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 (sets of braces).
Bags containing just 1 type of atomic 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 (x|-x-1).
Subtract (except: if x<y, then x-y results in zero.
Less than or equal to.
Times.
Compose 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.
Square (x) (Times (x,x)).
Coalesced repeat: (x (S F) take each S, op F, and repeat F(s) all along S)
Coalesced join: append together F(s), for each s∈S.
Coalesced replace: replace each element s∈S by F(s).
Compose three operations: S(F,G,H) = F o (G o H).
Compose three operations: S(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}, (N (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) | p,q,r are primes ∧ p+q=r}).