MASSACHUSSETS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

Working Paper 293 March 1987

Discovery Systems: From AM to CYRANO

Ken Haase

The emergence in 1976 of Doug Lenat's mathematical discovery program AM [Len76] [Len82a] was met with suprise and controversy; AM's performance seemed to bring the dream of super-intelligent machines to our doorstep, with amazingly simple methods to boot. However, the seeming promise of AM was not borne out: no generation of automated super-mathematicians appeared. Lenat's subsequent attempts (with his work on the Eurisko program) to explain and alleviate AM's problems were something of a novelty in Artificial Intelligence research; AI projects are usually 'let lie' after a brief moment in the limelight with a handful of examples. Lenat's work on Eurisko revealed certain constraints on the design of discovery programs; in particular, Lenat discovered that a close coupling of representation syntax and semantics is neccessary for a discovery program to prosper in a given domain. After Eurisko, my own work on the discovery program Cyrano has revealed more constraints on discovery processes in general in particular, work on Cyrano has revealed a requirement of 'closure' in concept formation. The concepts generated by a discovery program's concept formation component must be usable as inputs to that same concept formation component. Beginning with a theoretical analysis of AM's actual performance, this program presents a theory of discovery and goes on to present the implementation of an experiment — the CYRANO program — based on this theory. (This article is a preliminary version of an invited paper for the First International Symposium on Artificial Intelligence and Expert Systems, to be held in Berlin on May 18-22 1987.)

1 Introduction

The emergence in 1976 of Doug Lenat's mathematical discovery program AM [Len76] [Len82a] was met with suprise and controversy; AM's performance seemed to bring the dream of super-intelligent machines to our doorstep, with amazingly simple methods to boot. However, the seeming promise of AM was not borne out: no generation of automated super-mathematicians appeared. Lenat's subsequent attempts to explain and alleviate AM's problems [Len82b] [Len83b] [Len83a] [LB83] were something of a novelty in Artificial Intelligence research; AI projects are usually 'let lie' after a brief moment in the limelight with a handful of examples. The research described here both examines and extends Lenat's follow-up on the successes and failures of AM.

This paper is thus about discovery both in general and in particular. In general, it proposes a theory of discovery and learning and a set of constraints on the implementation of discovery programs. In particular, it is about the evolution and 'discovery' of this theory, based on the results and analyses of several discovery programs. In describing discovery in general, this paper is theoretical; in describing a particular discovery, this paper is methodological and descriptive.

The author's work with the discovery program CYRANO [Haa86b] — still ongoing — builds on Lenat's seminal work in discovery with both AM and Eurisko. The CYRANO program implements a theory which views discovery — and indeed, nearly all learning — as a process of forming new representational vocabulary from the empirical regularities and potential structure in an existing vocabulary. This new vocabulary — describing a world grounded in but abstracted from the previous vocabulary — is then used as a base for further analysis, discovery, and definition.

This image of the discovery process emerges from a detailed study of the successes and failures of AM, Eurisko, and early versions of CYRANO. In each of these programs, the construction and subsequent use of new representations forms a cycle of development or abstraction which must be closed if the discovery process is to sustain itself; if newly developed concepts or definitions are unusable as the basis for further discovery, the progress of the program will halt, stuck at a particular level of discovery or sophistication.

This view of discovery owes a significant debt to the images, metaphors, and genius of Piaget's constructive developmental psychology. Our programs — I argue — should learn like our children; ever growing into new ways of seeing and manipulating the world, beginning from a foundation of the worlds they already inhabit. When I speak of discovery, I include not only the realm of scientific or mathematical discovery, but the dozens of daily discoveries of a five-year old child or the hesitant steps of a freshman physics student in the footprints of millions before her. There is — from my perspective — a common thread to these experiences; they all arise from the effort to reshape or fill out an incompletely understood world by imposing or finding order in it. The theoretical question addressed in this paper is then: 'What must the structure of such structure-constructing programs be?' We begin to answer this in the next section by examining the structure of the 'cycle of discovery'.

2 The Cycle of Discovery

Figure 1 illustrates the discovery cycle introduced in the previous section. The central notion of the cycle of discovery is deceptively simple. Any discovery program — given a represented domain — works off of the

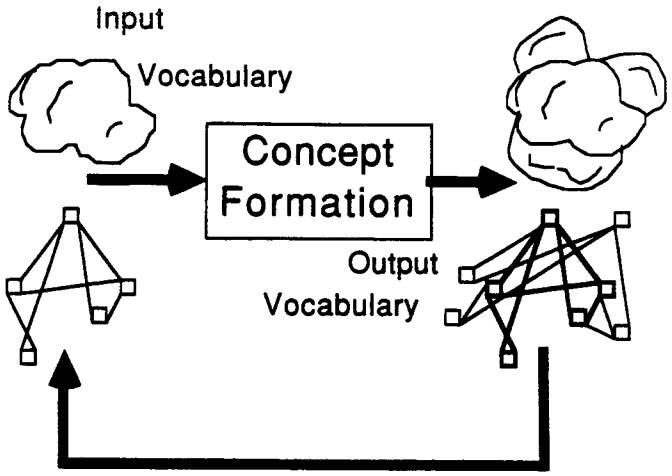

Figure 1. This depicts the cycle of discovery. A process of 'concept formation' acts on a given representation and — based on empirical properties and structural attributes — constructs a new representation which can be further processed by succeeding phases of concept formation.

empirical properties of that domain to produce definitions which combine domain concepts in some sensible manner. If we assume a finite flat (non-recursive) combinatorial vocabulary (ways of combining definitions to produce new objects), we have a space of new definitions which is some polynomial of the number of initial concepts. The order of this polynomial is the maximal arity of any combinator and, given a reasonably compact starting vocabulary, this space is relatively small and practically enumerable. The trivial new definitions derivable from a given vocabulary are — though not uninteresting — inherently limited. To go any further, the developed concepts of the program must be subject to combination and extension as well; the process of concept formation must operate on its own outputs as well as any user-provided inputs. This is the notion of the cycle of discovery.

When the space of concepts may feed upon itself, its size becomes exponential in the number of cycles the program is permitted to run. In order to control this exponential explosion, all discovery programs are *heuristic*; rather than enumerate entire spaces of concepts, they selectively enumerate subspaces of possible concepts. The subtitle of Lenat's thesis, "The Application of Heuristic Search to Discovery In Mathematics," captures this image of the discovery process. I believe, however, that this image is incomplete and that a return to the image of a cycle is appropriate.

In the characterization of discovery as search, heuristics enumerate subspaces of defined concepts; in the characterization of discovery as cycle, heuristics define new vocabularies from which further empirical analysis and concept formation may proceed. These 'vocabularies' are the terms in which new concepts and definitions will be cast; they define a 'flat' space for new concept definitions. In AM, Eurisko, and Cyrano, vocabularies are collections of categories (classes or types) and operations (between categories). Returning to the perspective of discovery as cycle, two important properties of the discovery process are brought to light: the *coherence* of vocabularies and the importance of closure in concept definition.

In general, a particular discovery constructs a new concept — or class of concepts — about which

a coterie of auxiliary concepts or operations define a new vocabulary to be explored and analyzed. This vocabulary is *coherent* in that it forms a structured and interconnected web of new definitions. While AM and Eurisko both produced vocabularies, their coherence is made explicit in the CYRANO program by the introduction of *abstraction functions* which map between vocabularies. The notion of coherent vocabularies and explicit abstractions between them suggest a knowledge-intensive way of controlling the search for new concepts; the development of new concepts is still heuristic — based on empirical analysis of base vocabularies — but is now organized in a way that makes the cycle of discovery explicit.

Even if the cycle of discovery is not made explicit in the structure of a program — as is largely the case in AM and Eurisko — it nonetheless exists in a distributed fashion; new representations are used to construct still newer representations. In order for this process to succeed, it must not 'get stuck' with representational vocabularies which cannot be further explored or analyzed. In the following section, I will demonstrate — with examples from AM, Eurisko, and Cyrano — how discovery programs can close the cycle of discovery to generate new concepts and definitions which *do* provide a basis for further discovery and definition. After describing several 'good' heuristics or mechanisms, I go on to describe how a discovery program — in particular, one component of AM — fails to close the cycle of discovery by generating concepts that are interesting but not represented in a way which enables further growth and development.

3 Closing The Cycle: Good Heuristics

In this section, I will describe a series of 'good' cycle-closing heuristics for discovery, extracted from the AM, Eurisko, and Cyrano. After describing several good heuristics, I describe a bad heuristic from AM which breaks the cycle, and argue that such heuristics contributed significantly to AM's eventual failure.

The first heuristic we look at is a simple category formation heuristic:

If some but not all examples of a concept C are also examples of a concept D
Then define the intersection $C \wedge D$ as those examples of C which are also examples of D.

This constructs concepts which are statistically significant and — of equal importance — can be used as new terms in other combinations. In fact, the same heuristic which created an intersection $C \wedge D$ can be applied to again to the new concept if another 'some but not most' regularity is noticed. Viewed as vocabulary formation, this is creating a new term about which other new terms (for instance, operations on C's applied solely to instances of the new term) can be defined. It is a special case of what I have called abstraction: abstraction by specialization.

AM used this heuristic to note that some, but not most, factorizations of numbers were pairs. It then defined the set of factorizations which were pairs and looked at what numbers factored into this set. The resulting concept, prime numbers, was one of AM's most exciting discoveries. Had AM merely noted the coincidence of factors and pairs (say, with a statistical annotation) the subsequent development of concepts which captured the regularity would not have occured, and the discovery of primes (and their use in hypotheses like Goldbach's conjecture) would have never been possible. The critical event at this point in AM's progress was the *formation* of the concept 'primes'.

Another case of abstraction by specialization is AM's domain restriction heuristic:

If an operation O has a domain D with an interesting specialization D'
Then define a specialization O' which is O restricted to operation on instances of D'

this constructs a new term (an operation) which can be either combined with other operations or used in defining new concepts which come from operations (for instance, the class determined by the range of the restricted operation).

AM used this heuristic to eventually propose Goldbach's conjecture that every even number is expressible as the sum of primes. It did this by restricting the addition operation from sets of numbers to sets of prime numbers (it had already defined and discovered prime numbers). It then noticed that the range of this new operation was exactly the even numbers; thus it proposed a general conjecture about such sums and even numbers.

The domain restriction heuristic implements — to some degree — the notion of 'coherent vocabularies' described above. However, rather than making the notion of distcint vocabularies explicit, the domain restriction heuristic forms a single 'vocabulary' loosely coupled together by notations of 'interstingness'. The term 'interesting' in the heuristic is neccessary because AM had no explicit notion of abstraction or the coherence of generated vocabularies; there was no way for AM to decide that terms involving a concept C were interesting, other than declaring C interesting. In Cyrano, the domain restriction heuristic is implemented by a general representation for abstraction which carries over operations (as well as other representational elements) through abstraction functions.

4 Breaking the Cycle: Bad Heuristics

An instance of a bad heuristic from AM is the so called 'CANONIZE' heuristic;[1] this is an example of a heuristic which fails to close the cycle of discovery, and as a result contributed to the eventual 'failure' of the program.

The CANONIZE heuristic as described by Lenat is:

> **CANONIZE** *is both an example of and a specialization of 'Operation'. It accepts two predicates* p_1 *and* p_2 *as arguments, both defined over a domain* $A \times A$*, where* p_1 *is a generalization of* p_2*. Canonize then tries to produce a "standard representation" for elements of A, in the following way. It creates an operation f from A into A, satisfying* $p_1(x, y) \longleftrightarrow p_2(f(x), f(y))$*. Then any item of the form* $f(x)$ *is called a canonical member of A. The set of canonical-A's is worth naming, and it is worth investigating the restriction sof various operations' domains and ranges to this set of canonical-As.*

In practice, the CANONIZE heuristic played a critical role in AM's progress, developing the representation which Lenat called 'Numbers' and lifting its speculations from the domain of set theory to the domain of number theory. Given a definition for LIST-EQUAL (provided by Lenat) and a definition for LIST-SAME-SIZE (generalized by AM from the definition of LIST-EQUAL), the CANONIZE heuristic produced a function which translated any two lists of the same size into two equal lists. This function converted each element of the lists into a unique element T; it's domain — called 'BAGS-OF-Ts' — captured as a

[1] This heuristic in fact should have been called, CANONICALIZE, but for reasons of pronouncability, the more saintly version was chosen.

representational term the notion of cardinal identity inherent in LIST-SAME-SIZE. The class 'BAGS-OF-Ts' was named 'Numbers' by Lenat and then used (by the domain restriction heuristic described above) to define operations like addition, multiplication, and so forth.

The CANONIZE heuristic in AM was only used once, for the leap from bags to numbers. But looking at its behaviour and implementation, we can see that this was foredoomed, for the way in which it worked failed to close the cycle of discovery and obsoleted itself upon its initial application.

Logically, what CANONIZE did was to notice that the SAME-SIZE operation partitioned the set of BAGS in a particular way (in fact, an equivalence partition) and defined an operation which produced a particular (in terms of OBJECT-EQUAL) representative of each partition.

From the point of view of closing the cycle of discovery, these new 'abstractions' were not proper concepts — defined equivalence classes — but were rather particular tokens which represented (particularly to the user) classes of objects. The cycle was not closed because it did not produce full-fledged concepts; rather it defined a specialization of BAGS which had interesting properties; the connection of these properties to the corresponding equivalence classes was provided by Lenat, when he named the concept 'Numbers'.

This, one could argue, is merely a philosophical complaint; if we imposed our assumptions on the program, it is just the same as though the program had grown to fill our assumptions. But a deeper problem arises from the manner in which CANONIZE worked internally. The mechanism of CANONIZE did not in fact recognize partitions in general, but only recognized a handful of partitions determined by trivial mutations on list structure: element permutation, addition, deletion, modification, etc. But the sorts of partitions CANONIZE could recognize was fixed initially and a generated concept (like Bags-Of-Ts) was 'structureless' once generated. The CANONIZE heuristic, by virtue of its implementation, immediately obsoleted itself with its first concept formation; it is no suprise that it was only used once.

Cyrano implements a version of CANONIZE which defines abstractions by producing actual equivalence classes (e.g. the class of all lists with five elements). We can imagine (and will see, when CYRANO makes sufficient progress) that this version of CANONIZE will notice partitions among other defined partitions/abstractions (e.g. modular numbers partitioned from the partitions defining numbers) or that it will discover partitions on composite structures which contain other defined abstractions (e.g. rational pairs or arbitrary vectors defined over natural numbers). However, the version of CANONIZE implemented by AM — impoverished as it is — is unable to notice such structure because its operation produces concepts it cannot itself deal with, breaking the cycle of discovery.

In the sections above, we described ways in which closure of the discovery cycle are assured in the AM, Eurisko, and Cyrano programs. Then we showed how one particular part of the AM program broke the cycle of discovery by implementing heuristics which produced definitions they themselves could not deal with, obsoleting themselves with their first application. In the next section, we discuss how CYRANO attempts to implement the model of the discovery process given above; explicitly implementing the notion of abstraction and using a uniform mechanism for defining new or given concepts from primitive components.

5 Cyrano: The Implementation

The considerations on discovery programs presented above arose from a detailed examination and partial reimplementation of the AM and Eurisko programs. After an initial reimplementation effort — an effort

staying very close to the original design of AM and Eurisko — the ideas above were formulated. My current research program is implementing a discovery program around the key points of the previous section. This section describes the implementation of that program.

The representation used by AM, Eurisko, and CYRANO-0[2] was a frame based representation language; in the case of Eurisko and CYRANO-0 it was a *representation language language* (RLL-1 [GL80] [Gre80] in the case of Eurisko; ARLO [Haa86a] in the case of CYRANO-0). For reasons detailed in [Haa87b], frame based RLL's were abandoned for the current implementation of CYRANO. Instead the representation language used is a type specification and inference language called TYPICAL and described in detail in [Haa87a]. TYPICAL is a combinator language for defining new predicates/types in a lattice of subsumption; when a new predicate/type is defined by combining existing types, TYPICAL makes a set of heuristic inferences about the relation of the new type to other types. Particularly, TYPICAL makes inferences about subsumption relations between types; whether satisfaction of one type neccessarily entails satisfaction of another.

All new concepts and definitions developed by CYRANO are either types in TYPICAL's lattice or mappings between types in the lattice. New types are defined by combining either existing types, existing mappings, or primitive SCHEME predicates or procedures. Such a uniform way of constructing new types from old is an important part of maintaining the closure of the discovery cycle. If concept formation is a module whose output/input feedback forms the discovery cycle, the common representation of the lattice provides a common interface at the incoming and outgoing edges of the module boundary.

The firing of heuristics is organized through this lattice; types in the lattice are annotated with daemons to apply to instances of the type. When a concept is given or created, all of these daemons are run, driving the proposal of hypotheses, the definition of 'related concepts,' and the formation of wholly new concepts. This control mechanism is detailed — as an application of TYPICAL — in [Haa87a].

In addition to representing new and given concepts and organizing control in Cyrano, the lattice is used to organize the confirmation of empiricial regularities. Most of the regularities noticed by CYRANO are expressed in terms of functions which generate example and counterexample spaces for the regularity. Cyrano's major activity consists of attempting — as a problem solver — to generate examples of these spaces. This process of confirmation is detailed in [Haa87a].

As mentioned above, the notion of abstraction is made explicit in CYRANO. An abstraction function is a function from a set of particulars (which are sometimes types) to a set of types which represent an *abstraction* of the domain of the function. The term abstraction is used loosely here; what is strictly meant by abstraction (in the implementation of CYRANO) is a mapping between two representational spaces. In the simple case mentioned above, abstraction by specialization, the mapping is an introjection; it is used particularly to 'scale down' vocabularies to subsets of the represented domain.

Another quite common abstraction function (used by CYRANO in place of AM's CANONIZE) is the PARTITION abstraction for a given relation. Each relation has two partition abstractions which map from the domain of the relation into a space of types/predicates represented in TYPICAL. The two abstractions are 'right' and 'left' partitions of the relation: the right partition of a relation '$\sim$' maps any element x into a type which is satisfied by some u only if $x \sim u$; the left partition maps any element x into a type which is

[2]This is the prototypical version of CYRANO implemented almost directly from Lenat's description of AM and Eurisko.

satisfied by some u only if $u \sim x$. If a relation is symmetric these partitions are the same; otherwise they are different.

An abstraction maps from one space into another and carries with it operations and relations in the original space. For instance, if APPEND is an operation on lists and the abstraction SAME-SIZE-PARTITION maps from list-space into number/cardinality-partition space, the operation which we call PLUS would be automatically generated as a carry over with the abstraction.

Where do abstractions come from? To begin, abstractions are the core of concept formation; each truly new concept has an accompanying abstraction. The problem of concept formation is a search problem in the space of abstractions; thus we have the two problems that appear immediately in search: how to generate possibilities and how to prune possibilities. Given classes of abstractions — like specialization to intersections or partitions of relations — the generation of possiblities is relatively straightforward. The second problem — of pruning possible abstractions — brings us to unknown lands, for the CYRANO program has not yet advanced far enough to give us insight into how to prune spaces of abstractions. But our current research brings us to speculate on possible ways of pruning these spaces.

One possible principle for pruning is based on the 'coherence of vocabularies' introduced above; we might prune from our search those abstractions which fail — heuristically — some criterion of coherence. But using such a criterion requires a precise notion of coherence which has not yet been advanced. Part of our current research program is an effort to acheive this precision; our suggestion is that each way of generating a new abstraction possesses a related method for judging the effectiveness of the abstraction.

One requirement of a coherent vocabulary is that the classifications it makes not be empty; that the classes and concepts it defines have both examples and non-examples. For if the concepts created cover nearly everything, they will tell us little more than we already know. One criterion along these lines, implemented in the current version of CYRANO, prunes partition abstractions based on the statistical properties of the produced partitions. In this implementation, a partition abstraction is declared and then studied — as an abstraction — by application to some random subset of the space it is partitioning. The resulting partitions are then analyzed to find their 'spread': how partition size compares to the size of the space partitioned. Relations which have huge spreads are not very useful as partition definers because they define concepts which have huge overlaps and fail to really 'distinguish' much. Of course, such a failure might suggest other abstractions (than partitions) which might be effective for the relation (for instance, point ordering abstractions or neighborhood groupings).

Again, these are merely speculations; the real results will come from the progress of the CYRANO program. And it is to this methodology of experimentation which we finally turn, to treat briefly the issue of analysis and reproducibility in Artificial Intelligence.

6 Methodology: From AM to Cyrano

Lenat's step of following up on the failures of AM is something of a methodological novelty in Artificial Intelligence. While the 'Future Work' section of research reports often point out weaknesses of the current research and directions for exploration, these suffer from two sorts of flaws. They are either implementational inadequacies which are attacked by a flurry of tool-building which loses the problem or they involve deep-seated issues which are at least as interesting (involving, distracting, etc) as the original issue.

In the first sort of case, energy is invested to produce nice (though this is always subject to argument) tools which make implementing the original program (or a slightly better one) easier. The problems of the program, however, are lost. In the second sort of case, the work focusses on some very difficult 'hard problem' and a variety of models and programs are tried as approaches to the 'hard problem.' In this case, the original problem is lost and with it, the issues (except for one) raised by it. Seldom does the researcher return — bearing a partial solution to the 'hard problem' — to try again and see what this partial solution will do. This step of Lenat's is the methodological novelty of his work. I believe that I am continuing this work with the development of the CYRANO program.

This is possible — at least in the domain of discovery — because of several factors. Two particular factors are a focus on *analysis* of the programs progress and a pragmatic attitude towards mechanisms and particular programs.

The focus on analysis of a program's progress was critical to both Lenat's development of Eurisko from AM and — as can be seen from the analyses earlier in the paper — the development from CYRANO from AM and Eurisko. This was possible in the domain of discovery because both the AM and Eurisko programs provided a wealth of examples. By the very nature of the vocabularies it develops and the cycle of discovery it implements, a discovery program is applied again and again to produce a wealth of generated examples. One way to think about discovery programs are that they are generators of examples of discovery! This twist of perspective actually turns out to be useful for considering introspection as a domain for automated discovery; by the same token it aids the human researcher in examining the discovery process implemented by his or her program.

The pragamatic attitude towards mechanisms and programs — the second reason discovery is a successful domain for AI by experimentation — reveals itself in the vocabulary used to describe our programs. We use terms like 'concept,' 'abstraction,' or 'experiment' quite loosely, leaving the implementation details and mechanisms as 'engineering problems'. While this leaves us floating over the pit of McDermott's 'natural stupidity' criticism[McD81], our defense is that we do not believe that our programs' 'concepts' or 'abstractions' are concepts or abstractions in the broadest sense. But we use these terms — vauge and flaky as they are — as a 'task level' or 'knowledge level' [New82] characterization of our problem. And as researchers in the area of discovery, we are always returning to this level for deployment and analysis of our interim solutions.

Pragmatically, having such a cavalier attitude towards mechanisms and programs requires a certain competence in the maintenance of large programs which may not be universal. As in the natural sciences, the role of good 'experimentalists' emerges into Artificial Intelligence in the form of programmers who are effective at keeping track of and easily modifying large and complicated systems. But these 'experimentalists' are able to see different problems and appreciate different biases, since 'how to do the experiment' is both more binding (it may be part of how they think) and less binding (when thinking, they have more latitude for experimentation) on their theory making.

The careful and well-thought-out criticisms of Lenat in [RH83] may be a criticism of experimentalists by theorists. Such criticism is important, as is criticism in the opposite direction, but the importance of both perspectives is also crucial to progress if we are to remain honest to those notions that keep us from being particularly — on the one hand — logicians or philosophers or — on the other hand — engineers and programmers.

7 References

[GL80] Russell Greiner and Douglas Lenat. A representation language language. In *AAAI-80*, American Association for Artificial Intelligence, 1980.

[Gre80] Russell Greiner. *A Representation Language Language.* Stanford HPP Report HPP-80-9, Computer Science Department, Stanford University, 1980.

[Haa86a] Kenneth W. Haase. *ARLO: Another Representation Language Offer.* Technical Report 901, MIT Artificial Intelligence Laboratory, 1986.

[Haa86b] Kenneth W. Haase. Discovery systems. In *ECAI-86*, Europeran Conference on Artificial Intelligence, 1986.

[Haa87a] Kenneth W. Haase. *TYPICAL: An Implemented Approach to Type Specification and Inference.* Technical Report 922, MIT Artificial Intelligence Laboratory, 1987. (in preparation).

[Haa87b] Kenneth W. Haase. *Why Representation Language Languages are No Good.* Memo 921, MIT Artificial Intelligence Laboratory, 1987.

[LB83] Douglas B. Lenat and Jon S. Brown. Why AM and Eurisko Appear to Work. *Artificial Intelligence*, 23, 1983.

[Len76] Douglas B. Lenat. *AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search.* PhD thesis, Stanford University, 1976.

[Len82a] Douglas B. Lenat. AM: Discovery in Mathematics as Heuristic Search. In Douglas B. Lenat and Randall Davis, editors, *Knowledge Based Systems in Artificial Intelligence*, McGraw-Hill Book Company, 1982. Several appendices of examples were trimmed from the original version of the thesis in this book version.

[Len82b] Douglas B. Lenat. The Nature of Heuristics, Part I. *Artificial Intelligence*, 19, 1982.

[Len83a] Douglas B. Lenat. Eurisko: A program which learns new heuristics and domain concepts. *Artificial Intelligence*, 21, 1983.

[Len83b] Douglas B. Lenat. Theory Formation by Heuristic Search. *Artificial Intelligence*, 21, 1983.

[McD81] Drew McDermott. Artificial intelligence meets natural stupidity. In John Haugeland, editor, *Mind Design*, Bradford Books, 1981.

[New82] Allen Newell. Aaai president's address. *AI Magazine*, 1982.

[RH83] G.D. Ritchie and F.K. Hanna. AM: A Case Study in AI Methodology. *Artificial Intelligence*, 23, 1983.