# HEURETICS: THEORETICAL AND EXPERIMENTAL STUDY OF HEURISTIC RULES

Douglas B. Lenat
Heuristic Programming Project
Stanford University

Builders of expert rule-based systems [Barr 81] [Feigenbaum 77] [Hayes-Roth *et al.* 82] attribute the impressive performance of their programs to the corpus of knowledge they embody: a large network of facts to provide breadth of scope, and a large array of informal judgmental rules (heuristics) which guide the system toward plausible paths to follow and away from implausible ones.

Yet what is the nature of heuristics? What is the source of their power? How do they interrelate; i.e., how can/should a large corpus of heuristic rules be organized? How do heuristics originate and evolve?

"Heuretics" is the study of heuristics, with an eye toward answering questions such as those. Two case studies, the AM and EURISKO programs, have led to some tentative Heuretics hypotheses, a dozen of which are presented in this paper. Our aim is to stimulate future research in this field.

**Hypothesis 1: Heuretics is a *bona fide* field of knowledge, and merits investigation by AI.** We speak of Heuretics as a *field of knowledge* because (as we sketched above) it has some more or less well agreed-upon objects of study, some motivation for studying such objects, and some central questions about the nature of such objects.

But to rate as a *field of knowledge*, as a science, Heuretics must also possess some accepted methods for investigating its many questions. We hypothesize the adequacy of the standard empirical inquiry paradigm which dominates AI research; i.e,, test hypotheses about heuristics by constructing -- and studying -- computer programs which use heuristics and which try to find new ones.

**Hypothesis 2: Heuristic rules have three primary uses: to prune away implausible "moves" (actions, alternatives, etc.), to propose plausible ones, and to serve as data for the induction of new heuristic rules.** The first of these, using a heuristic to prune a search, is the one most heavily studied by earlier workers in the field (Michie, Nilsson, Gaschnig, etc. See, for example, [Gaschnig 77] and the references he cites.). The second use, plausible move generation, is the source of power exploited by AM [Lenat 79]. It has the character of *learning by discovery*. In the third and final case, the entities being learned are not new domain concepts, but new heuristics.

**Hypothesis 3: Heuristics can act as "plausible move generators", to guide an explorer -- be he human or machine -- toward valuable new concepts worthy of attention.** This is one of the three roles for a heuristic rule, as noted in Hyp. 2. Here we illustrate how that works. Consider, as an example, the heuristic H1:

`H1: if function f takes a pair of A's as arguments, then define and study g(x) =df f(x,x).`

That is, given a binary function f, it's often worth taking the time and energy to see what happens when f's arguments coincide. If f is multiplication, this new function turns out to be squaring; if f is addition, g is doubling. If f is union or intersection, g is the identity function; if f is subtraction or exclusive-or, g is identically zero. Thus we see how two useful concepts (squaring, doubling) and four fundamental conjectures might be discovered by a researcher employing this simple heuristic.

Application of H1 is not limited to mathematics of course; one can think of Compile(x,x) [i.e., optimizing compilers written inefficiently in the language they compile, and then processed by themselves]; Kill(x,x) [i.e., suicide]; Ponder(x,x) [i.e., self-awareness]; and even Apply(x,x) [i.e., the activity we are now engaging in]. This hypothesis was suggested by work with AM, and has been confirmed in several domains by recent experiments with EURISKO.

When EURISKO was applied to the task of generating interesting three-dimensional VLSI designs, it already possessed a heuristic that suggested augmenting any design, to make it more symmetric. When this was applied to the two-dimensional primitive device (the gate: see Fig. 1), it led to the very symmetric device in

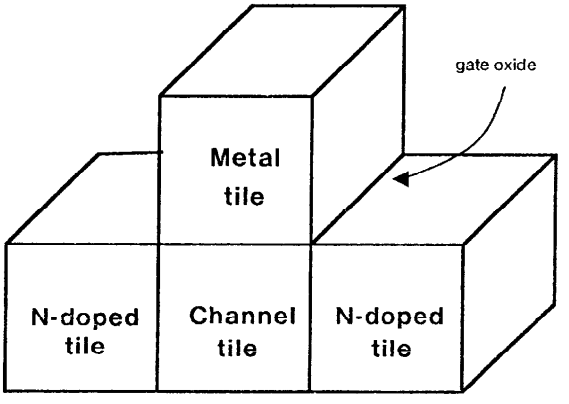


Figure 1. The standard MOS gate. Side view. The Channel tile is intrinsic channel; i.e., it can serve between two positively (p-) doped regions or between two negatively (n-) doped regions.

Figure 2 below, a device which simultaneously computes NAND and OR, and which is the fundamental building block of most of our latest 3-D VLSI chip designs. Symmetry is often more than merely aesthetic; here, it led to a device which tesselates (packs) three-space, and can be compactly stacked up into memory cells, PLAs, etc. James Gibbons, a pioneer in the techniques of building high-rise chips by recrystallization of thin silicon films [Gibbons 80], has recently succeeded in fabricating these devices for us.

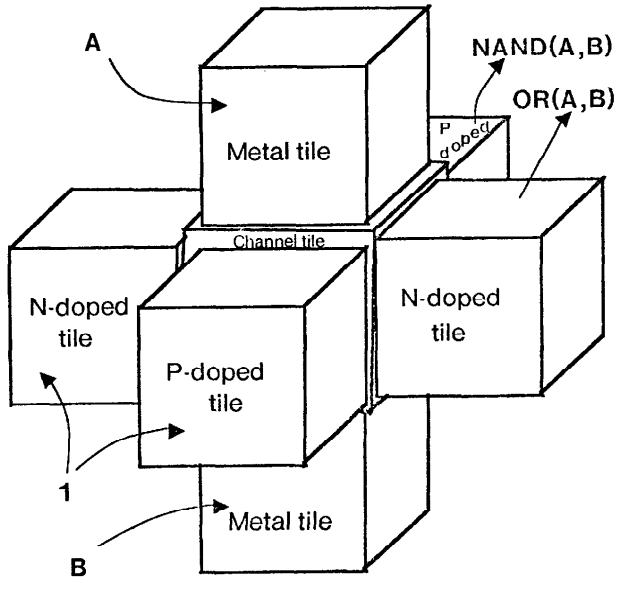


Figure 2 A symmetrized, three-dimensional extension of the gate from Figure 1. The central tile is intrinsic channel, coated with gate oxide on both its top and bottom surfaces.

**Hypothesis 4: The same methodology enables a body of heuristic rules to monitor, modify, and enlarge itself.** Each heuristic in EURISKO is represented as a full-fledged concept, a frame with dozens of slots, each with a relatively small, succint value. This "parameterization" of the space of heuristics enables a corpus of heuristics to apply to itself. For instance, a heuristic that says "If a concept is using inordinate resources and achieving very little, then put it out of its misery (drop the ExpectedWorth to zero and stop using it)" can apply to mathematical functions and can also apply to heuristics. The heuristic "If a concept is sometimes useful but often not, then specialize it in many ways" actually applied to itself successfully, in one of the early runs of EURISKO, because *it* was sometimes useful but often not.

One of EURISKO's tasks is the design of naval ships, and entire fleets, conforming to a large set of rules and formulae (GDW's "Trillion Credit Squadron" game). A simulator was easily built, and this enabled a kind of heuristically-guided evolution of fleets, with the simulator providing the needed natural selection function. As each battle was fought, specific designs were given credit and blame, and abstractions of these formed simple design heuristics (e.g., "Heavy armor is more important than agility"). Eventually, EURISKO generalized these into a very abstract heursitic: For the values of most parameters, when designing a TCS fleet, the best value will usually be a nearly -- but not quite -- extreme value. This was noticed at the level of designing an individual ship (nearly heaviest armor, nearly as many types of weapons as legally allowed, nearly as small weapons as possible, etc.), and turned out to apply to the design of the fleet as a whole (nearly uniform, nearly minimal speed, etc.)

The fleet designed by this process won the national Origins tournament in TCS last July 4 (seven-round, single elimination). Partly as a result of the counter-intuitive loopholes exploited by that design, there were numerous rules changes in effect for the local tournament held in February, 1982. It is significant that EURISKO spent much less time developing a fleet (yes, it won again) for that set of rules, as most of its general heuristics were still valid, even though the particular designs it came up with were quite different (e.g., a ship with no offensive ability was a useful adjunct to its July, 1981 fleet; a ship with no *defense* was useful for the February, 1982 tournament; the old fleet had practically no ships with large weapons, wheras the new fleet had practically no ships *without* such weapons). EURISKO's design task was made much easier by the use of the new heuristics it synthesized last summer.

**Hypothesis 5: Heuristics are compiled hindsight, and draw their power from the various kinds of regularity and continuity in the world.** If an action A was (or would have been) useful in situation S, then it is likely that actions similar to A will be useful in the future in situations similar to S. I.e., if we could somehow actually compute the utility of obeying a heuristic, then that function -- APPROPRIATENESS(Action,Situation) -- would be continuous in both variables. One useful exercise is to consider the graph of APPROPRIATENESS values for a fixed action, varying over the situations in which it might be applied.

For example, consider graphing the utility of the heuristic "If it's April already and you haven't gotten your taxes done, then consider going to a commercial tax-preparer". The value of this advice varies as a function of many situation-dependent variables, such as your income (see Fig. 3). If you earned below $12k, it might be better to do it yourself; if you earned above $28k, it might be better to get an accountant.

The language of graphs of functions is now at our disposal, an attractive metaphor within which to discuss such processes as specializing a heuristic, using multiple heuristics, and measuring attributes of a heuristic's performance. For instance, we were led to ponder the significance of the *slope* of the curve as it intersects the x-axis. This reflects how crucial it is to determine true -- rather than approximate -- relevance of the heuristic. If the slope is steep, the effects of obeying the heuristic when it isn't quite relevant could be

catastrophic. If the slope is mild, spending a great amount of time determining true relevance is a waste. We cannot automatically construct the graph of a heuristic's utility, but it is easy to infer the magnitude of its slope near the x-intercept from several cases of its accidental misuse. EURISKO presently uses this empirical technique to estimate this quantity for each heuristic. That value in turn guides the rule interpreter in deciding how much time to spend evaluating the IF-part of the rule before trying to apply the THEN-part.

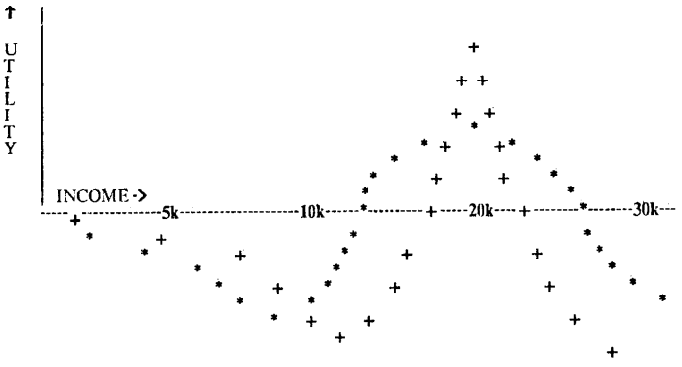


Figure 3. The utility of "...go to a commerical income-tax-preparer" (*) and "...go to John Smith at the Palo Alto AxaTax office" (+).

As an example of specializing a heuristic, consider what happens as we change the THEN-part of the above heuristic into "... then go see John Smith at the Palo Alto AxaTax office". His forte' may be individuals whose income is around $20k, and he may be *worse* than his colleages for taxpayers earning about $12k, or about $28k. See the graph marked "+" in Fig. 3. If we specialize the THEN-part of a heuristic, it typically has higher utility but only over a narrower domain. Notice the area under the curve appears to be remain roughly constant; this is a geometric interpretation of the tradeoff between generality and power of heuristic rules.

**Hypothesis 6: Generalizing and specializing a heuristic often leads to a pathologically extreme new one, but some of those are useful nevertheless.** By examining the graphs in Figure 3 above, one can generate a list of possible bugs that may occur when the actions (THEN-part) of a heuristic are specialized. First, the domain of relevance of the new one may be so narrow that it is merely a spike, a delta function. This is what happens when a general heuristic is replaced by a table of specific values. A more common bug occurs when one of the heuristics is completely dominated by the other. For example, "Smack a vu-graph projector if it makes noise" has much narrower domain, but no higher utility, than the more general heuristic "Smack a device if it's acting up". Thus, the area under the curve is greatly diminished, but no benefit (narrow, high peak) accrues.

While the last paragraph warned of some extreme bad cases of specializing the THEN- part of a heuristic, there are some extreme good cases which frequently occur. The utility (y-) axis may have some absolute desirable point along it (e.g., some guarantee of correctness or efficiency), and by specializing the heuristic enough, its utility may exceed that threshhold (albeit over a narrow range of tasks). In such a case, the way we *qualitatively* value that heuristic may alter; e.g., we may term it "algorithmic" or "real-time" or "0-tax".

So some of the most useful constructs in computer science can be viewed as pathological cases of heuristics. Algorithms are seen to be heuristics which are so powerful that guarantees can be made about their use. Tables of values are seen to be heuristics whose domain is a set of measure zero.

**Hypothesis 7: The graph of "all the world's heuristics" is surprisingly shallow.** One can take a specific heuristic and generalize it gradually, in all posible ways, until all the generalizations collapse into weak methods. By carrying out this activity over and over again, for heuristics from various fields, we can imagine building up a graph that would approach -- in the limit -- "the graph of all heuristics." This might be a useful technique, then, for investigating the space of heuristics, getting at its structure.

A preliminary analysis (using AM's 243 heuristics) led us to expect the tree to be of maximum depth about 50, though most of AM's heuristics were turned into weak methods after only about a dozen generalization steps. Next, with the help of Woody Bledsoe and Herbert Simon, we analyzed this partial tree of AM's heuristics, examining the power of the rules therein. It soon became apparent that most generalizations $H_{genl}$ were just as powerful than the heuristic rule(s) $H_{spec}$ beneath them! In such cases, the specific rule(s) can be eliminated from the tree. The resulting tree had depth of only 4, and is thus incredibly shallow and bushy. We observed that all but the top couple and bottom couple levels of its tree could be eliminated with no ill effects.

This elevates the top levels (the weak methods) to a special status, and likewise the bottom levels (the domain-specific rules). It may shed some light on the successes of two radically opposed philosophies of heuristic search: the cognitive science and knowledge engineering. It also dooms most attempts to create a new heuristic by specializing (moving downwards in that tree): the new heuristics synthesized that way are almost sure to be no more powerful than the ones you started with. This explains theoretically a finding we reached empirically using EURISKO: generalization and analogy -- *not* specialization -- are the most powerful ways to discover new heuristics.

One solution to this "shallow-tree problem" is the realization that there are numerous ways in which one heuristic can relate to another: abstraction-of, applies-more-widely-than, possibly-triggers, easier-to-teach-than, etc. If there is *any* such relation for which $H_{new}$ has some higher power than $H_{old}$, then it's worth keeping $H_{new}$ around. E.g., special cases of Maxwell's equations and Thorp's Blackjack tens-count systems are

important, as they are easier to teach and use than the general versions.

**Hypothesis 8: Even though the world is often discontinuous, we usually cannot do any better than rely upon heuristics that presume continuity (see Hyp. 5).** There are many possible measures of APPROPRIATENESS (e.g., efficiency, low down-side risk, comprehensibility), and many dimensions along which Situations can vary (e.g., difficulty, time, importance, subject matter). Compounding this is the nonlinearity of the Situation space along most of these dimensions. Thus the "zero$^{th}$ order theory" espoused in Hyps. 5-7 is only a metaphor.

Yet it is too attractive, too close to what human experts actually do, to reject out of hand. It can be extended into a "first order theory":

It is frequently useful to *behave* as though the zero$^{th}$ order theory were true. That is, one acts as if the function APPROPRIATENESS(`Action,Situation`) existed, were computable, continuous, and time-invariant..

To give an example: the current situation may appear similar to ones in which it was cost-effective to skip to the Conclusions section of the paper. Even though you can't be sure that that's an appropriate action to take now, it may be useful for you to behave as though the world is that continuous, to take that action anyway. If you do so, you're following a heuristic. [To not tempt the reader to follow that heuristic, this paper has no Conclusions section.] That heuristic's guidance is only as good as the generalization process you used in deciding the situation was similar (e.g., would you apply it to all articles? to all articles written by Lenat?) The world has of course changed in innumerable ways since the formation of your heuristics about paper-reading. You cannot monitor even a small fraction of the changes in the world; you cope (i.e., solve the Frame Problem) by relying on extant heuristics and revising them as -- but usually only as -- they fail you.

**Hypothesis 9: The interrelations among a set of heuristics -- and the internal structure of a single one -- can and *should* be quite complex.** Six years ago, the AM program was constructed as an experiment in learning by discovery. Its source of power was 243 heuristics, rules which guided it toward fruitful topics of investigation, toward profitable experiments to perform, toward plausible hypotheses and definitions. Its ultimate limitation apparently was due to its inability to discover new, powerful, domain-specific heuristics for the various new fields it uncovered. At that time, it seemed straight-forward to simply add "Heuristics" as one more field in which to let AM explore, observe, define, and develop.

That task -- learning new heuristics by discovery -- turned out to be much more difficult than was realized initially, and we have just now achieved some successes at it. Along the way, it became clearer why AM had succeeded in the first place, and why it was so difficult to use the same paradigm to discover new heuristics. In essence, AM was an automatic programming system, whose primitive actions were modifications to pieces of Lisp code, predicates which represented the characteristic functions of various math concepts. See [Green *et al.*, 74] for background on this style of code synthesis and modification. It was only because of the deep relationship between Lisp and Mathematics that these operations (loop unwinding, recursion elimination, composition, argument elimination, function substitution, etc.) which were basic Lisp mutators also turned out to yield a high "hit rate" of viable, useful new math concepts when applied to previously-known, useful math concepts. For instance, AM took a piece of Lisp code which determined whether or not two list structures were *equal*, lopped off one recursive call, and (see Fig. 4) wound up with a new Lisp predicate that returned True iff its arguments had *the same length*. But no such deep relationship existed between Lisp and Heuristics: when the basic automatic programming operators were applied to viable, useful heuristics, they almost always produced useless (often worse than useless) new heuristic rules.

```
PRED1: λ (x,y)
         (COND ((EQ x y) T)
               ((OR (NULL x) (NULL y)) NIL)
               (T (AND (PRED1 (CAR x) (CAR y))
                       (PRED1 (CDR x) (CDR y)))))

PRED2: λ (x,y)
           (COND ((EQ x y) T)
                 ((OR (NULL x) (NULL y)) NIL)
                 (T (PRED2 (CDR x) (CDR y))))
```

Figure 4. AM was given PRED1, the characteristic function for the math concept "list-equality". By lopping off one recursive call, it created PRED2, which computes "lists having the same length".

Over the past six years, we have constructed a new language in which the statement of heuristics is more natural and compact. The vocabulary includes many types of conditions (If-there-are-enough-resources, If-we-recently-worked-on), actions (Then-conjecture, Then-add-to-agenda), and non-executable attributes that record descriptive information about the heuristic (Average-running-time, Origin). Instead of writing two large lumps of Lisp code to represent the heuristic (If and Then), one spreads the same information out across many dozens of "slots". A heuristic in EURISKO is now -- like a math concept always was in AM -- a collection of about twenty or more slots, each filled with a line or two worth of code. By employing this new language, the old property that AM satisfied *fortuitously* is once again satisfied: the primitive syntactic operators usually now produce meaningful semantic variants of what they operate on. The ties to the foundations of Heuretics have been engineered into the syntax and vocabulary of the new language, partly by design and partly by evolution, much as McCarthy engineered ties to the foundations of Mathematics into Lisp.

The EURISKO program employs this language to represent hundreds of heuristics. It explores eight task domains: design of naval fleets, elementary set theory and number theory, Lisp programming, biological evolution, games in general, the design of three-dimensional VLSI devices, the discovery of heuristics

which help the system discover heuristics, and the discovery of appropriate new types of "slots" in each domain. From 200 to 500 concepts from each domain (objects, operators, etc.) were also supplied initially. In each domain, many new concepts and designs, and a few heuristics, were indeed discovered mechanically.

**Hypothesis 10: Some very general heuristics can guide the search for new domain-dependent heuristics.** Even though two domains may appear disparate, the same heuristics may be equally powerful in coping with them. This one, e.g., is useful in almost all fields:

```
IF you're about to use or prove something about C,
THEN first make certain that C has some examples,
        i.e., ensure that C is not vacuous.
```

Even if the *heuristics* for the two domains seem disparate, the paths which were followed in getting the powerful heuristics of the field may be similar.

**Hypothesis 11: As with any field of human endeavor, Heuretics is accumulating a corpus of informal judgmental knowledge -- heuristics.** In this case, the heuristics are about Heuretics; they guide the heuretician in extracting heuristics from experts, in deciding when the existing corpus of heuristics needs to be augmented, in representing heuristics within knowledge bases, in evaluating the worth of a heuristic, in troubleshooting a program built around a large collection of heuristic rules, etc. Some examples are:

```
The best way to extract a new heuristic is to have
the expert watch the existing program err, and ask
him/her to help track down the "bug".

When generalizing a heuristic, don't replace the
central "AND" connective of the If-Potentially-
Relevant slot by "OR"; the result is indeed more
general, but likely to be very over-generalized.

New heuristics should emerge much less often than
new domain concepts.

If a heuristic takes up a lot of room, there's
probably a useful new slot or two that can be
defined, to shorten that (and other) heuristics.
```

**Hypothesis 12: Domains which are unexplored, internally formalizable, combinatorially immense, and highly structured are ideal for studying Heuretics.** EURISKO found many new concepts and heuristics in the TCS game and the 3D VLSI design task because *any* discoveries in those domains were almost certain to be new: the domains were practically unexplored by human beings. Breaking new ground is often easier than scouring old ground for neglected gems.

What other criteria make a task well-suited to automated discovery? A second one is that there must be a way to simulate or directly carry out experiments. Third, the "search space" should be too immense for other methods to work. No human should be able to manually search the same space the program is walking around in. Fourth, the task domain must be rich in structure, including *heuristic structure*. There should be many objects and operators, many kinds of objects and kinds of operators. Hopefully they will be related hierarchically and in other ways. Complexity of the domain raises the utility of plausible, inexact reasoning, hence the need for heuristic guidance. By "heuristic structure" we mean the presence of many heuristics, and implicitly the absence of any known efficient algorithms to replace them. For instance, theorem proving in *propositional* calculus is a poor task domain for Heuretics research, as it admits only a few heuristics, and they are already well known.

Finally, one of the most crucial requirements is that of an adequate representation. If the language or representation employed is not well matched to the domain objects and operators, the heuristics that do exist will be long and awkwardly stated, and the discovery of new ones in that representation may be nearly impossible. An example of this was the painful development of EURISKO, which began with a small vocabulary of slots for describing heuristics (If, Then), and was forced (in order to obtain reasonable performance) to evolve two orders of magnitude more kinds of slots that heuristics could have, some of them domain-dependent..

The purpose of this paper has not been to convincingly argue each point. Rather, we hope to stimulate research in a new area -- Heuretics -- and to that end have indicated a spectrum of questions, apparent regularities, and issues which are worth investigating further.

## Acknowledgments

Productive discussions with John Seely Brown, Bruce Buchanan, Bill Clancey, Ed Feigenbaum, Johan deKleer, George Polya, Herb Simon, and Mike Williams have heavily influenced this work. EURISKO is written in -- and relies upon -- RLL. The 3D VLSI work is in collaboration with Bert Sutherland of SSA and Jim Gibbons of Stanford. Finally, I wish to thank XEROX PARC's CIS and Stanford University's HPP for providing superb environments (intellectual, physical, and computational) in which to work. Financial support has been provided by ONR (N00014-80-C-0609) and XEROX PARC.

## References

Barr, Avron, and Edward A. Feigenbaum, eds., *Handbook of AI*, Volume II, William Kaufman, Los Altos, 1981.

Feigenbaum, Edward A., "The Art of Artificial Intelligence", *Proc. Fifth IJCAI*, Cambridge, Mass., August, 1977, p. 1014.

Gaschnig, John, "Exactly How Good Are Heuristics?: Toward a Realistic Predictive Theory of Best-First Search", *Proc. Fifth IJCAI*, Cambridge, Mass., August, 1977.

Gibbons, James, and K. F. Lee, "One-Gate-Wide CMOS Inverter on Laser-Recrystallized Polysilicon", *IEEE Electron Device Letters*, EDL-1, 6, June, 1980.

Green, Cordell, Richard Waldinger, David Barstow, Robert Elschlager, Douglas Lenat, Brian McCune, David Shaw, and Louis Steinberg, *Progress Report on Program Understanding Systems*, STAN-CS-74-444, AI Lab, Stanford, August, 1974.

Hayes-Roth, Frederick, Donald Waterman, and Douglas Lenat (eds.), *Building Expert Systems*, proceedings of the 1980 San Diego workshop in expert systems, to appear 1982.

Lenat, Douglas B., "On Automated Scientific Theory Formation: A Case Study Using the AM Program," in (Hayes et al, eds.) *Machine Intell. 9*, NY: Halstead Press, 1979, pp. 251-283.

Lenat, Douglas B., "The Nature of Heuristics", *J. Artificial Intelligence*, to appear Fall, 1982.

Polya, G., *How to Solve It*, Princeton University Press, 1945.