A.I. AND THE LEARNING OF MATHEMATICS

Edwina L. Rissland*
Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003

Technical Report 83-40

A.I. AND THE LEARNING OF MATHEMATICS:

A TUTORIAL SAMPLING

Edwina L. Rissland*
Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003

Prepared for the Conference on Instructional Implications of Research on Mathematical Problem Solving, San Diego State University, June 3-5, 1983.

*Preparation of this paper was supported in part by grants SED 80-19328 and IST-8017343 of the National Science Foundation. Any opinions, conclusions or recommendations expressed are those of the author and do not necessarily reflect the views of the National Science Foundation.

INTRODUCTION

In this paper, I discuss studies of mathematics made from an A.I. (Artificial Intelligence) point of view. There is a spectrum of work in this area that ranges from studies whose purpose is solely to understand better the learning of mathematics to studies whose purpose is to perform a mathematical task without human involvement or consideration. Such a spectrum reflects the purposes of A.I. researchers themselves: those who use A.I. as an approach to study cognition and understand it better to those who use A.I. to build systems to do tasks without making claims that the manner in which the programs perform says anything about the way humans perform the task (see, e.g., [Samuel 1983; Schank 1983]).

In this paper, I divide the work to be discussed into the following categories:

1. A.I. conceptual analyses of mathematical tasks
2. A.I. process models to explain mathematical behaviors
3. A.I. programs to do mathematics somewhat like a mathematician
4. A.I. programs to do mathematics without claims of similarity to mathematicians

This paper by no means covers all the papers of relevance in A.I., Cognitive Science, and Mathematics Education. Rather it is a sampling of work which I consider interesting and indicative of what can be done in each of the categories. It is a sampler to give the reader an idea of what has been done already and to help a researcher outside of A.I. find entry points into the literature.

Some of the programs discussed are not new; they have been selected because they are "classics". Some of the work discussed is not really A.I.; it is included because it is the sort of study necessary before one can build an A.I. program — they are what I call one's "epistemological homework".

I have not tackled the vast literature on problem solving per se; it is too extensive to review well in a paper of this scope and it is not necessarily directed at the concerns of this workshop, namely, mathematics and education. I also have not ventured into areas related to mathematics like physics and computer programming; one would be well-advised to take a look at the work done in these areas by researchers such as Larkin, deKleer, Simon in physics and Soloway and colleagues in programming. I will not say much about programs in the fourth category. These programs are not particularly relevant to this audience.

Throughout this paper the reader will find various terms from A.I. For a discussion of such basics of A.I., see the Handbook of Artifical Intelligence [Barr and Feigenbaum 1981, 1982; Cohen and Feigenbaum 1982].

Outline of Rest of Paper

The topics and research to be discussed are:

I. A.I. conceptual analyses of mathematical tasks

1. Davis on Mathematical Understanding
2. Matz's work on High School Algebra
3. Schoenfeld on Heuristics
4. Rissland on Understanding Understanding Mathematics

II. A.I. Process Models for Mathematical Tasks

1. Bundy's work on Equation Solving
2. Brown and Burton's work on BUGGY
3. VanLehn's work on Repairs to Bugs
4. Rissland's work on Constrained Example Generation

III. A.I. Programs to do Mathematics Somewhat like a Mathematician

1. Slagle's SAINT program
2. Lenat's AM program
3. Selfridge's COUNT program
4. Mitchell's LEX program

I. A.I. CONCEPTUAL ANALYSES OF MATHEMATICAL TASKS

1. Davis' Work on Mathematical Understanding

Bob Davis and his co-workers at the Curriculum Laboratory at the University of Illinois have amassed an extraordinary body of material on children learning mathematics ranging from arithmetic to calculus, for example [Davis et al. 1978; Davis and McKnight 1980]. Their studies over the past ten years have involved extensive, thorough and sensitive protocol analyses of students working problems. They have concentrated on giving an elucidation of skills by attention to student mistakes and misconceptions. In their explanations, Davis et al. have concentrated on discussing the mathematics involved rather than specifying or building a detailed computational model. In their discussions of the processes and knowledge, they have used ideas, like "frames" and "subgoals", from A.I.

They have not pushed their work in the direction of developing a conceptual framework for mathematics; rather they have probed deeply into the mathematical tasks themselves and used, when appropriate, A.I. ideas of others. Their work offers an extensive base on which others can build; see, for example, the discussion of Matz below.

To give the reader an idea of Davis work, we present an example typical of his project's work taken from his recent final report [Davis et al. 1982]. It is a discussion of subtraction taken from a discussion arguing for the need for careful analysis of skills and not just superficial consideration of right and wrong answers a la "drill and practice".

"One of our studies [Davis and McKnight 1980] dealt with a third-grade girl, Marcia, who subtracted

$$\begin{array}{r} 7,\ 0\ 0\ 2 \\ -\ 2\ 5 \\ \hline \end{array}$$

by writing

$$\begin{array}{l} 5 \\ \cancel{6} \\ \cancel{7},\ \smile\ 0\ 2 \\ \quad\ \ -\ 2\ 5 \\ \hline 5,\ 0\ 8\ 7 \end{array}$$

Marcia was convinced that she had performed the subtraction correctly. ...What does this have to do with <u>understanding</u>?

....What has made the remediation so difficult in Marcia's case is that she believes

1. She has learned the subtraction algorithm carefully and well (and she has, provided there are no zeroes in "inside" columns in the minuend);
2. She always gets correct answers by using this algorithm (and she does — again, provided there are no zeroes in the "inside" columns in the minuend);
3. She is using the <u>same</u> algorithm for

$$\begin{array}{r} 7,\ 0\ 0\ 2 \\ -\ 2\ 5 \\ \hline \end{array}$$

that she uses for, say,

$$\begin{array}{r} 1,\ 9\ 8\ 5 \\ -\ 2\ 9\ 6 \\ \hline \end{array}$$

It is, of course, this third belief that causes the trouble. But, unfortunately, one cannot really say whether Marcia is correct, or not, in this belief. There are two possible rules that she <u>might</u> be using:

a) When necessary, "borrow" from the next digit on the left (in the minuend);

or else

b) When necessary, "borrow" from the nearest non-zero digit on the left (in the minuend).

No case had previously arisen that would distinguish between these two rules. (Indeed, the theory of "knowledge" which underlies our work suggests that probably Marcia had not formulated her "rule" so precisely that such a distinction could be described.)

Davis in his discussion places great emphasis on the problem of retrieval and matching in understanding. Again his project has gathered the kind of evidence that has and can be capitalized on in A.I. work. For instance, he discusses the valid and invalid retrieval of the axiom ("the zero product principle") "$A \times B=0 \Longrightarrow [A=0 \vee B=0]$" and its mis-generalization as "$A \times B=k \Longrightarrow [A=k \vee B=k]$", [Davis et al. 1982] at pp. 39, and 94, 95. This is exactly the sort of mathematical behavior that Matz describes in terms closer to those of A.I. In his discussion of retrieval, Davis suggests that the problem Marcia has is related to an incorrect, or rather an incomplete, retrieval of relevant knowledge.

Davis is making links with A.I. by using their conceptual constructs to describe what he sees in his work, for instance, he uses the notions of "slots" and "frames" to address the retrieval problem. He uses "procedures" and "sub-procedures" to describe the heirarchical nature of plans and skills like those involved in synthetic division. I chose the excerpts on subtraction and the zero product principle because they enter the work of Matz and van Lehn described below.

Davis also uses the ideas of "planning" and "critics", which are central in A.I., to describe what he sees in his students like Marcia. In particular, Marcia lacks, or is not using, the "size" critic for numbers that says that if one has about seven thousand dollars and spends twenty-five dollars, one should NOT end up with about five thousand dollars. (An A.I. critic or "demon" is a process that is always watching what is going on and when it sees something of interest, it "shouts".)

2. Matz's work on High School Algebra

Matz discusses how systematic errors in high school algebra problem solving are the result of resonable, although unsuccessful, attempts to adapt known procedures to a new situation by applying what she calls "extrapolation" techniques [Matz 1980; 1982].

Matz discusses that many common errors arise from one of two processes:

1. inappropriate use of a known rule as is in a new situation;
2. incorrect adaptation of a known rule to solve a new problem.

In particular with regard to the second class of error, she is interested in:

1. errors generated by an incorrect choice of an extrapolation technique;
2. errors reflecting an impoverished base knowledge;
3. errors arising during the execution of a procedure.

Particularly interesting is her discussion of extrapolation technique errors of generalization and linear decomposition. In the class of linearity errors are those errors resulting from overgeneralizing a distribution rule. These are typified by:

SQRT(A+B) = SQRT(A) + SQRT(B)

In such errors a composite algebraic expression is decomposed linearly by distributing its top-most operator. The following table is taken from her paper [1982]:

TABLE 1
Generalized distribution

Correct

$$A(B+C)=AB+AC$$
$$A(B-C)=AB-AC$$
$$\frac{1}{A}(B-C)=\frac{1}{A}(B)+\frac{1}{A}(C) \quad \text{equivalently,} \quad \frac{B+C}{A}=\frac{B}{A}+\frac{C}{A}$$
$$(AB)^2=A^2B^2 \quad \text{more generally,} \quad (AB)^n=A^nB^n$$
$$\sqrt(AB)=\sqrt A*\sqrt B \quad \text{more generally,} \quad (AB)^{1/n}=(A)^{1/n}(B)^{1/n}$$

Incorrect

$$\sqrt(A+B)\Rightarrow\sqrt A+\sqrt B$$
$$(A+B)^2\Rightarrow A^2+B^2$$
$$A(BC)\Rightarrow AB*AC$$
$$\frac{A}{B+C}\Rightarrow\frac{A}{B}+\frac{A}{C}$$
$$2^{a+b}\Rightarrow 2^a+2^b$$
$$2^{ab}\Rightarrow 2^a2^b$$

She generalizes these errors into three schemas, one of which is:

$$\square(X\blacktriangle Y) \implies \square X \blacktriangle \square Y.$$

It is based on the past experience that the distributive law worked successfully in past problems, so why not use it again. This is exactly the sort of retrieval problem that Davis is discussing. Incorrect retrieval and application of this schema results in mistakes typically involving square roots and powers.

Another common way to generalize a schema is to replace specific constants by variables as in the "zero product principle" described above in Davis' work. This is based on the typically valid assumption that the numbers involved in a procedure are incidental. Unfortunately, 0 is the well-known counter-example to this heuristic. Consider the following problem:

$$(X-3)(X-4) = 0$$
$$(X-3) = 0 \text{ or } (X-4)=0$$
$$X=3 \text{ or } X=4$$

generalized to:

$$(X-3)(X-4) = 7$$
$$(X-3) = 7 \text{ or } (X-4) = 7$$
$$X=10 \text{ or } X=11$$

It is an excellent example of a good strategy going awry. It is exactly the kind of strategy one would build into an A.I. learning program; however such a program would need a means (for instance, through "critics" or "generate and test" methodologies) to prune away false generalizations. Nevertheless, it would be the right kind of thing to try.

Matz's work is an example of research into the cognitive aspects of mathematics which seeks explanations in terms of A.I. concepts such as rules and procedures.

3. <u>Schoenfeld's work on Heuristics</u>

Alan Schoenfeld has devoted much of his research to discussing heuristics and their role in problem solving, for instance [Schoenfeld 1978]. He has been particularly interested in whether such high-level strategies can be taught to college undergraduates [Schoenfeld 1980]. His work, while not pushed to the level of description favored by A.I. researchers nor explicitly couched in A.I. terms, is an example of the kind of in-depth discussion needed to explicate a topic, heuristics, which A.I. researchers often assume explained by its mere mention. Heuristics are used heavily in A.I. work (for instance, in expert systems like MYCIN) and play a very key role in the work of Slagle and Lenat, described below in Section III. For these reasons, I include Schoenfeld's work in this section.

In his 1980 MAA article Schoenfeld poses the questions:

1. "Can we accurately describe the strategies used by 'expert' mathematicians to solve problems?"
and
2. "Can we teach students to use those strategies?"

He answers in the affirmative, as I and others like Davis would. He discusses the complexity of using a heuristic strategy like the oft-cited "find an analogous problem" heuristic and the inherent difficulties of carrying it out: which analogous problem, by what analogy, in what representation? Even the heuristic "If there is an integer parameter, look for an inductive argument" can be difficult to implement, especially for students, who often operate at the "syntactic" and not "semantic" level of understanding. Such heuristic strategies are really labels for a collection of more detailed procedural and descriptive knowledge and further (sub-)strategies. Recognition of this complexity is often missing in problem-solving discussions, even those of that great master Polya [1973].

In his 1980 article, Schoenfeld outlines a problem-solving strategy which uses modules to perform:

problem analysis
argument design
problem exploration
solution implementation
solution verification

He presents a flowchart-like schematic outline of his problem-solving strategy (Fig. 2, p. 800) and describes aspects of each of these components. Some of the "modules" in his schema could be forced to a descriptive level which would be implementable; some need much more specification.

More recently [Schoenfeld 1983a, b], he has been interested in exploring issues in the analysis of protocols and describing problem-solving behavior, especially its "control" component. He distinguishes three types of knowledge ingredients of problem-solving:

1. resources -- typically domain-specific knowledge such as facts and algorithms, routine procedures and heuristics, representations and other knowledge possessed by the individual which can be brought to bear on the problem at hand;

2. control -- planning, monitoring, assessment, "metacognitive" acts, and other ingredients related to the selection and implementation of tactical resources;

3. belief systems -- about self, the environment, topic and mathematics, which influence an individual's behavior.

These are elaborated in his 1983 AERA paper [Schoenfeld 1983b]. This paper contains a nice example (p. 8) of problem-solving which not only illustrates his points but also provides an example of how specific examples (like "reference" examples) enter into the problem-solving process.

A laudatory aspect of Schoenfeld's work is the inquiry into the nature of strategies and control-level knowledge. Too often such high-level knowledge is glossed-over in favor of tactical or algorithmic knowledge because it is easier to grapple with, in the sense of being more describable or implementable because of its step-by-step focus. Yet it is strategic level knowledge -- or control -- that is critical to learning both in people and machines; this theme will be apparent in Section III.

4. Rissland's work on Understanding Understanding Mathematics

Rissland, in her 1977 Ph. D. thesis, sought to elucidate how one understands a mathematical theory like eigenvalues or continuity. She posed the question, "What is it that I know when I understand a body of mathematics well?". The answer is couched in terms of a mapping-out of the knowledge in a scheme of "spaces", which are semantic networks of frame-like items linked together through various sorts of relations [Rissland 1978a, b, 1980]. The description of the conceptual framework relies heavily on computer science notions like data base items and pointer structures.

An application of this work, never implemented, was to have been an interactive environment for the professional mathematician to explore and retrieve information in a theory, such as real analysis, and an environment built on the professional's to help a neophyte learn _how_ to explore and understand.

Rissland's elucidation of understanding contains the following main ingredients:

1. _spaces_ of _items_ and _relations_, such as _Examples-space_, _Results-space_ and _Concepts-space_;

    1. _items_ are strongly bound clusters of information: for instance, the statement of a theorem, its name, its proof, a diagram, an evaluation of its importance, and remarks on its limitations and generality;

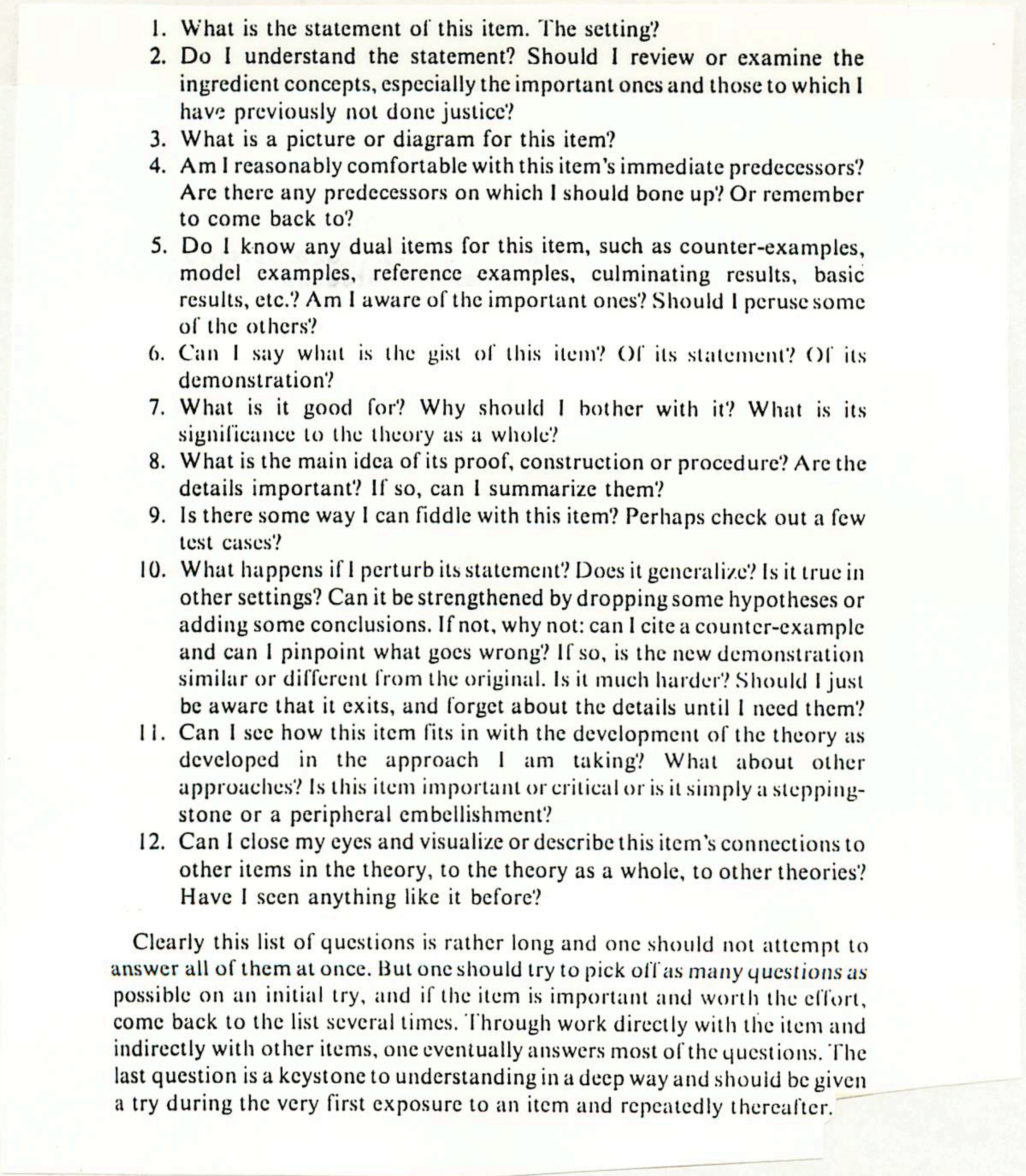
    2. _spaces_ are sets of similar types of items related in similar ways: for instance, proved results and their logical dependencies constitute _Results-space_.

2. a taxonomy of _epistemological classes_ of items based on their role in teaching, learning and understanding, and _worth ratings_ of items based on importance;

3. detailed laying out of <u>item-frames</u> for each of the principle type of item (example, result, concept);

4. detailed discussion of links between items: <u>in-space</u> links between items within a space (like <u>constructional</u> <u>derivation</u> between example items) and <u>across-space</u> ("dual") links;

"Understanding" involves enriching one's store of knowledge, particularly inter-frame and inter-space connections. The following is a list of questions which "prompt and probe" understanding [Rissland 1978b]:



1. What is the statement of this item. The setting?
2. Do I understand the statement? Should I review or examine the ingredient concepts, especially the important ones and those to which I have previously not done justice?
3. What is a picture or diagram for this item?
4. Am I reasonably comfortable with this item's immediate predecessors? Are there any predecessors on which I should bone up? Or remember to come back to?
5. Do I know any dual items for this item, such as counter-examples, model examples, reference examples, culminating results, basic results, etc.? Am I aware of the important ones? Should I peruse some of the others?
6. Can I say what is the gist of this item? Of its statement? Of its demonstration?
7. What is it good for? Why should I bother with it? What is its significance to the theory as a whole?
8. What is the main idea of its proof, construction or procedure? Are the details important? If so, can I summarize them?
9. Is there some way I can fiddle with this item? Perhaps check out a few test cases?
10. What happens if I perturb its statement? Does it generalize? Is it true in other settings? Can it be strengthened by dropping some hypotheses or adding some conclusions. If not, why not: can I cite a counter-example and can I pinpoint what goes wrong? If so, is the new demonstration similar or different from the original. Is it much harder? Should I just be aware that it exits, and forget about the details until I need them?
11. Can I see how this item fits in with the development of the theory as developed in the approach I am taking? What about other approaches? Is this item important or critical or is it simply a stepping-stone or a peripheral embellishment?
12. Can I close my eyes and visualize or describe this item's connections to other items in the theory, to the theory as a whole, to other theories? Have I seen anything like it before?

Clearly this list of questions is rather long and one should not attempt to answer all of them at once. But one should try to pick off as many questions as possible on an initial try, and if the item is important and worth the effort, come back to the list several times. Through work directly with the item and indirectly with other items, one eventually answers most of the questions. The last question is a keystone to understanding in a deep way and should be given a try during the very first exposure to an item and repeatedly thereafter.

In essence the idea is that items, which include results, concepts, examples, goals and strategies, are cohesive clusters of information and that there are important relations between them. Each space of items plus relations describes a different aspect of knowledge. Another important point is that one can taxonomize items on the basis of how they are used in learning, doing, teaching and understanding mathematics. The following describes these ideas as they relate to the "examples-side" of mathemtics.

An example, by which is meant a specific situation, case or experience, is comprised of many aspects or pieces of information: a name, taggings and annotations as to epistemological class and importance, lists of pointers to other examples from which it is constructed and to whose construction it contributes, the process of how it is constructed, a schematic or diagram, pointers to items like definitions and theorems in other spaces, statements of what the example is good for or how it can be misleading, sources of further information about the example. Examples are linked through the relation of constructional derivation of how one example is built from others. Examples plus this relation constitute an Examples-space.

When one considers the different effects and uses examples can have with respect to teaching, learning and understanding, one can distinguish different "epistemological" classes. There are similar analyses for results and concepts.

It is important to recognize that not all examples serve the same function. For instance, expert teachers and learners know that certain perspicuous ("start-up") examples provide easy access to a new topic, that some ("reference") examples are quite standard and make good illustrations, and that some examples are anomalous and don't seem to fit into one's understanding. Thus, one can develop a taxonomy:

(a) start-up examples: perspicuous, easily understood and easily presented cases;
(b) reference examples: standard, ubiquitous cases;
(c) counter examples: limiting, falsifying cases;
(d) model examples: general, paradigmatic cases;
(e) anomalous examples: exceptions and pathologies.

Start-up examples are simple, easy to understand and explain cases. They are particularly useful when one is learning or explaining a domain for the first time. Such examples can be generated with minimal reference to other examples; thus one can say they are structurally uncomplicated. A good start-up example is often "projective" in the sense that it is indicative of the general case and that what one learns about it can be "lifted" to more complex examples.

Reference examples are examples that one refers to over and over again. They are "textbook cases" which are widely applicable throughout a domain and thus provide a common point of reference through which many concepts, results and other items in the domain are (indirectly) linked together.

Counter-examples are examples that refute or limit. They are typically used to sharpen distinctions between concepts and to refine theorems or conjectures. They are essential to the process of "proofs and refutations", described beautifully by Imre Lakatos [1976].

Model examples are examples that are paradigmatic and generic. They suggest and summarize expectations and default assumptions about the general case. Thus, they are like "templates" or Minsky's "frames".

Anomalous examples are examples that do not seem to fit into one's knowledge of the domain, and yet they seem important. They are "funny" cases that nag at one's understanding. Sometimes resolving where they fit leads to a new level of understanding.

An example of applying this classification scheme for an introductory study of continuity from the domain of real function theory might classify: the function f(x)=x as a start-up example; f(x)=x**2, f(x)=e**x as reference examples; f(x)=1/x as a counter-example; "f(x) with no gaps or breaks" as a model example; and f(x)= sin(1/x) as an anomalous example. The first example, f(x)=x, is also a reference example (the "identity" function). Thus, such a classsification need not be exclusive. The anomaly sin(1/x) will most likely become a favorite counter-example as one understands that a function can fail to be continuous in at least two ways, that is, by having gaps and breaks and by failing to settle down to a limit. Thus, such a classification is not static. Increased understanding will of course lead to qualifications on the above model of a continuous function, although it will still serve to summarize one's expectations.

II. A.I. PROCESS MODELS FOR MATHEMATICAL TASKS

1. Bundy's work on Equation Solving

Alan Bundy, of the University of Edinburgh, in connection with his work on automatic theorem proving, looked at the processes involved in solving algebraic equations. He described high-level processes which account for the line-to-line transitions in solving equations for an unknown [Bundy 1975, Bundy and Silver 1981, Borning and Bundy 1981].

Bundy calls his high-level processes "strategies"; they are

1. isolation
2. collection
3. attraction
4. cancellation
5. removing nasty function symbols

The first three constitute what he calls the "basic method".

Bundy's analysis is an example of research pursued for A.I. purposes, here automatic theorem proving, which is very relevant to the study and explication of mathematical expertise in humans. It provides a detailed description in "hard" computational terms of a skill and provides an opportunity for use in mathematics education since his descriptors and procedures are readily understandable. One could easily talk to a class working on equation solving using his ideas of strategies and basic methods. In fact, I would recommend exactly that.

An example to illustrate his analysis is the following:

```
1.  ln(x+1) + ln(x-1) = 3
                                .......(i)
2.  ln[(x+1)(x-1)] = 3
                                .......(ii)
3.  ln(x**2 - 1)  = 3
                                .......(iii)
4.  x**2 - 1  = e**3
```

.......(iv)

5. x**2 = e**3 + 1

.......(v)

6. x = SQRT(e**3+1) or x = -SQRT(e**3+1)

In his analysis of the above sequence of equations, Bundy is interested in explicating what happens in between the lines, that is, the transitions and transformations used. He points out that many of the solution steps are not shown and that one is using more than the usual axioms of the real numbers (this last observation stems from his concern about automatic theorem proving in which the prover must have access to such knowledge). For instance, any prover must have a wealth of knowledge on how to perform <u>simplification</u>. (As Bundy discusses there is no universal standard of simplification: what is simpler in one context is not necessarily so in another). Regarding this point, he considers step (ii), the transition from line 2 to line 3:

> "[W]e will speculate that the axiom
> (u+v)(u-v) = u**2 - v**2
> was used, producing
> ln(x**2 -1**2) = 3
> and that the simplification step, from this equation to line 3. was omitted.

Bundy's analysis of the above goes as follows:

> "We look first at the end of the solution, lines 3 to 6. In line 3, for the first time the equation contains only a single occurrence of the unknown, x. From here on the solution is straightforward. The next three steps consist of stripping away the functions surrounding this single occurrence of x until it is left isolated on the left hand side of the equation. Each step consists of identifying the outermost (or dominant) function symbol on the left-hand-side, recovering the axiom which will remove it from the left-hand-side and insert its inverse on the right-hand-side, and then applying this axiom. We will call the strategy which guides these three steps <u>isolation</u>... It can be regarded as the basis of nearly all work in equation solving
>
> ..As with simplification, mathematicians often omit isolation steps from their written protocols, crowding as many as three or four steps into the transition from one line to another.

> We next look at line 2:
>
> 2. ln(x+1)(x-1) = 3
>
> This contains 2 occurrences of x, so isolation is not applicable. However, we can see step (ii) as <u>preparing for isolation</u> (our emphasis), by achieving a reduction in the number of occurrences of x from 2 to 1. This is done by applying the identity:
>
> (u+v)(u-v) = u**2 - v**2.
>
> This is an example of our second strategy, which we call <u>collection</u>, namely, when there is more than one occurrence of the unknown, try to find an axiom which will collect occurrences together.
>
> Finally we look at line 1 and step (i).
>
> 1. ln(x+1) + ln(x-1) = 3
>
> The two occurrences of x cannot be immediately collected, presumably because a suitable axiom was not stored. We can however <u>prepare</u> ..<u>for collection</u> (our emphasis) by moving them closer together, so that more identities will match the term containing them both. This is what happens in step (i)....The strategies of moving occurrences closer together to increase their chances of collection, we call <u>attraction</u>.
>
> When the above three strategies are combined in this way we call the resulting equation solving strategy, <u>the basic method</u>.

[Bundy 1975 at page 17].

Bundy now proceeds to turn describe his three strategies in computational terms. This involves, for instance, providing "harder" definitions of "term" and the notion of "closer".

For instance, in his discussion on collection (p. 21):

> "If the two occurrences are on the same side of the equation this will be a term called the <u>containing term</u>, otherwise it will be the whole equation. We deal with the former case first. The containing term must now be replaced by another term containing one occurrence of the unknown, say x, so we must look for an axiom, say A, which will do this. We can easily build up a description, which A must obey.
>
> (i) A must be an identity, i.e., an expression of the form:
> s1 = s2 or B --> s1 = s2, where B is called its precondition.
>
> (ii) One of the variables, say u, mus occur either
> (a) twice in s1 and once in s2
> or (b) twice in s2 and once in s1
> without loss of generality we will assume case (a).
>
> (iii) s1 must match the containing term, with u being instantiated to x.
>
> ....
>
> If A obeys parts i) and ii) (a) of the above description we will say that it is <u>useful to collection, left to right, relative to u</u>. If A obeys parts i) and ii) (b) we will say that it is <u>useful to collection, right to left, relative to u</u>.

...[E].g.,
$\sin 2u = 2.\sin u.\cos u$
is useful to collection, right to left, relative to u.
$(u+v)(u-v) = u**2 - v**2$
is useful to collection, left to right, relative to u.

Bundy forces all three strategies, isolation, collection and attraction, to this degree of specificity in computational terms. He then gives two more examples of equation solving, one involving trigonometric functions, the other a general quadratic, to illustrate his strategies. The latter is included as an appendix to this paper.

Bundy's last major point concerns the removal of "nasty function symbols" in equation solving. Examples of nasty symbols in this context are radicals and inverse trig functions like arcsin. Bundy provides a hierarchy of niceness/nastiness. In removing nasty symbols, he identifies and describes three major strategies:

1. cancellation
2. collection
3. inversion

The end result of such an analysis in the case of a rationalizaton goes as follows:

1. $\sqrt{5x-25} - \sqrt{x-1} = 2$

........(i)

2. $\sqrt{5x-25} = 2 + \sqrt{x-1}$ isolation

........(ii)

3. $5x-25 = (2+\sqrt{x-1})**2$ inversion

........(iii)

4. $5x-25 = 2**2 + 2.2\sqrt{x-1} + (\sqrt{x-1})**2$ cancellation

........(iv)

5. $5x-25 = 4 + 4.\sqrt{x-1} + (x-1)$

........(v)

6. $4x-28 = 4.\sqrt{x-1}$

........(vi)

7. $\sqrt{x-1} = (x-7)**2$ isolation

........(vii) inversion

8. $x-1 = (x-7)**2$

Eliminating radicals, the process of "rationalization", is a paradigmatic example of removing a nasty function symbol. Reproducing Bundy's analysis here would take too much space. One comment on these processes is that they now apply to function symbols rather than unknowns. For instance, one tries to get occurrences of a function and its inverse, like sin and arcsin, closer together.

2. Brown and Burton's work on BUGGY

To account for student errors in simple procedural skills like subtraction Brown and Burton [1978; Burton 1982] proposed "the Buggy model". In that model, student errors are seen as symptoms of a "bug", that is, a discrete modification to a correct procedural skill. For example, the bug "0-n=n" accounts for the errors in both of the following subtraction problems:

```
 500      312
 -65     -243
 ---      ---
 565      149
```

The bug dictates that when the top digit in a column is 0, write the bottom digit as the answer.

BUGGY is a computerized game based on the diagnostic interactions between a student and a teacher. The computer plays the part of a student who has a bug. The challenge to the user is to find the bug; BUGGY presents examples of the "student's" incorrectly done homework problems. The user/diagnostician describes the bug. The conjectured diagnoses is then tested with more examples of problems done by the student.

The main thrust of the BUGGY work is thus the diagnoses of bugs with the assistance of a computer program (which can be seen as a gaming environment). This leads (hopefully) to an improvement in the skill of the diagnostician, like a student teacher. BUGGY therefore can be used to explore such high-levels skills as hypothesis formation, strategic knowledge, debugging, and theory testing through examples. Brown and Burton feel that their BUGGY work provides a language for both teachers and students to talk about their work and that this is a good thing. This tenet also underlies the work of Rissland and Bundy; Papert elsewhere has often addressed this issue [Papert 1980].

The BUGGY work was motiviated in part in response to the often incorrect assumption that student errors are "random" or that students do not follow procedures very well. The BUGGY work suggests quite the opposite: students are often too faithful to procedures. Such conclusions have also been reached by Davis and his colleagues, and Piaget, long beforehand.

Often the manifestations of a "buggy" procedure do not permit easy diagnoses; it is easy to see something is awry but much harder to say exactly what. For instance, Brown and Burton challenge their readers to diagnose the bug in the following series of subtraction problems:

$$\begin{array}{r}7\\+8\\\hline 15\end{array}\quad\begin{array}{r}9\\+5\\\hline 14\end{array}\quad\begin{array}{r}8\\+3\\\hline 11\end{array}\quad\begin{array}{r}6\\+7\\\hline 13\end{array}\quad\begin{array}{r}8\\+8\\\hline 16\end{array}\quad\begin{array}{r}9\\+9\\\hline 18\end{array}\quad\begin{array}{r}17\\+8\\\hline 25\end{array}\quad\begin{array}{r}19\\+4\\\hline 23\end{array}\quad\begin{array}{r}87\\+93\\\hline 11\end{array}$$

$$\begin{array}{r}365\\+574\\\hline 819\end{array}\quad\begin{array}{r}679\\+794\\\hline 111\end{array}\quad\begin{array}{r}923\\+481\\\hline 114\end{array}\quad\begin{array}{r}27,493\\+1,509\\\hline 28,991\end{array}\quad\begin{array}{r}797\\+48,632\\\hline 48,119\end{array}$$

The bug here is that every time there is a carry, the student is simply writing down the carry and forgetting about the units digit. This student also had this same bug in his multiplication skill. Thus bugs can piggyback on subprocedures into higher level procedures and cause bugs in these as well. Remediation of the higher-level skill would require remediation of the lower one much like the debugging of procedures in computer programming often requires the debugging of subprocedures.

The theme here is the breakdown of skills into simpler and more primitive sub-skills. For instance, in ordinary additio^n primitives might be recognizing a digit, writing a digit, etc. The analysis of a skill in terms of such primitives is a standard A.I. approach; it is seen in the work of Selfridge and Briars and Larkin discussed below.

BUGGY makes its diagnoses by picking a bug from approximately one hundred known bugs by comparing the student's answers with the output of each bug run on the test problems. The initial hypothesis set contains any bug that explains at least one of the student's bugs on the problem as a whole (as compared to an error in a single column). This initial hypothesis set is reduced by finding and removing primitive bugs that are completely subsumed by other primitive bugs.

The information BUGGY uses is summarized in "bug comparison" tables such as:

```
     8    99   353   633    81  4769   257  6523   103  7315  1039   705 10038 10060  7001
     3    79   342   221    17     0   161  1280    64  6536    44     9  4319    98    94
     -  ----  ----  ----  ----  ----  ----  ----  ----  ----  ----  ----  ----  ----  ----
     5    20    11   412    64  4769    96  5243    39   779   995   696  5719  9962  6907
Student answers:
     -     -     -     -    98     -   418     -   169   738  1095   706 14319 10078  7097

*FORGET/BORROW/OVER/BLANKS:
     *     *     *     *     !     *     !     *   139     !   ***   *** 15719 10062  7007

*STOPS/BORROW/AT/ZERO:
     *     *     *     *     !     *     !     *    49     !   ***   ***  6719 10062  7017

*DIFF/0-N=N:
     *     *     *     *     !     *     !     *     !   839     !     !   ***  9978     !

*ADD/INSTEADOF/SUB:
    11   178   695   854   ***     *   ***  7803   167 13851  1083   714 14357 10158  7095
```

The problems with the correct answers appear at the top. Student answers appear on the next line with a "-" indicating a student correct answer. Each of the remaining line contain the name of a bug and information regarding the running of the subtraction procedure with that bug. For these lines, a "***" indicates the bug predicts the student's incorrect answer; "*" indicates both the student and the bugged procedure got the problem right; a "!" indicates the bugged procedure gave the correct answer but the student did not; a number entry is the answer the bugged procedure would get when it is different from both the student and the correct answer. Thus "*" and "***" are confirming evidence for the student having a bug -- both are producing the same results -- and "!" is disconfirming evidence.

Using such evidence, BUGGY then rates each bug according to how well it explains the student bug. The system does this by means of a "symptom" vector that contains information such as the number of "***"s.

In their work, Brown and Burton discuss many of the subtleties that may occur, such as: (1) some bugs are caused by performance lapses (e.g., the student copies a number wrong); (2) some bugs are due to errors in subskills (e.g., errors in standard subtraction fact table like 10-2=3); (3) bugs can act in concert and thus one bug can hide the existence of another (e.g., the "smaller-from-larger" bug will hide any bug in the borrowing procedure).

3. VanLehn's Repair Theory

Given that bugs in procedural skills do occur, a natural question to ask is "How do they arise?" Matz has given some answers with her analysis of over-generalization. VanLehn and Brown offer another explanation by generating "bug stories" that describe how a particular bug could arise [Brown and VanLehn 1980]. Their work, which deals with subtraction errors, is a natural extension of Brown and Burton's.

The Repair Theory explanation for the origin of bugs is as follows:

"When one has unsuccessfully applied a procedure to a given problem, one attempts a repair. The need to make a repair is often triggered by a procedure (or sub-procedure) reaching an impasse by which is meant a state in which it cannot execute. One tries to remedy the impasses by applying repair heuristics, of which the following are considered:
1. skip
2. quit (the procedure)
3. swap vertically (the subtrahend and minuend entries)
4. dememorize

VanLehn and Brown experiment with the principles of their theory by taking a production-rule representation of a standard school subtraction algorithm, performing deletions on it to create buggy procedures, and then running the repair theory. They are in no way claiming that the incorrect procedures are generated by one forgetting or deleting lines from a correct procedures; these incorrect procedures are just a means to study the repairs.

To give the reader an idea of Repair Theory, we reproduce Figure 1 from the VanLehn and Brown article (p. 387):

The syntax is:

```
Goal (goal's arguments)  Satisfaction Condition: goal's satisfaction condition
    label:    {rule's conditions} --->              rule's action
    other rules for achieving the goal...
```

The rules for the version of subtraction used in this paper are:

```
Sub ()  Satisfaction Condition: TRUE
    L1:     {} --->                        (ColSequence RightmostTopCell
                                             RightmostBottomCell RightmostAnswerCell)
ColSequence (TC BC AC)  Satisfaction Condition: (Blank? (Next TC))
    L2:     {} --->                        (SubCol TC BC AC)
    L3:     {} --->                        (ColSequence (Next TC) (Next BC) (Next AC))
SubCol (TC BC AC)  Satisfaction Condition: (NOT (Blank? AC))
    L4:     {(Blank? BC)} --->             (WriteAns TC AC)
    L5:     {(Less? TC BC)} --->           (Borrow TC)
    L6:     {} --->                        (Diff TC BC AC)
Borrow (TC)  Satisfaction Condition: FALSE
    L7:     {} --->                        (BorrowFrom (Next TC))
    L8:     {} --->                        (Add10 TC)
BorrowFrom (TC)  Satisfaction Condition: TRUE
    L9:     {(Zero? TC)} --->              (BorrowFromZero TC)
    L10:    {} --->                        (Decr TC)
BorrowFromZero (TC)  Satisfaction Condition: FALSE
    L11:    {} --->                        (Write9 TC)
    L12:    {} --->                        (BorrowFrom (Next TC) )
```

TC, BC and AC are variables. Their names are mneumonic for their contents, which happen to be the top, bottom and answer cells of a column.

The primitive actions and their associated preconditions are listed below. All of their arguments are cells. The actions expecting digits in certain arguments have a precondition that those cells not be blank.

Diff -- Subtracts the digit contained in its second argument from the digit contained in its first argument and writes the result in the third argument. The second argument can not be larger than the first argument.

Decr -- Subtracts one from the digit contained in its argument and writes the result back in the same cell. The input digit must be larger than zero.

WriteAns -- Writes the digit contained in its first argument in its second argument.

Add10 -- Adds ten to the digit contained in its argument and writes the result back in the same cell.

Write9 -- Writes a nine in its argument. The cell can not be blank originally.

Figure 1. A GAO graph for a standard version of subtraction

"When a rule is deleted, its sister rules will often be executed in its place, which frequently leads to an impasse. For example, when L4 of Figure 1 has been deleted, and the procedure is run on the problem

```
 27
 -4
 --
```

an impasses is reached in the tens column because the interpreter choses L6, the only rule that applies given that L4 is gone. Running L6 results in calling the primitive action Diff. Diff takes a column difference by taking the difference of its first two arguments' contents and writing the result in the cell pointed to by the third argument. But Diff has a precondition that neither of its arguments be blank. Since this precondition is violated when Diff is called on the tens column, the procedure is at an impasse. This impasse can be repaired in a variety of ways. For example, the procedure could simply do nothing instead of take the column difference (the "no-op" repair heuristic). Control would return from Diff, and ultimately the procedure would terminate normally leaving 3 as the answer. This way of repairing the impasse generates the bug Quit-When-Bottom-Blank.

They describe several bugs generated by repairs to procedures with incorrectly deleted lines. For instance, if L5 (from Figure 1) is deleted.

This is the rule that says to borrow when the top digit is too small. If L5 is deleted, then L6, the rule for processing ordinary columns, will be executed on every column, including larger from smaller (LFS) columns where one ought to borrow. LFS columns violate a precondition of the Diff (the action called by L6), namely that the first input number be larger than the second input number. This precondition violation is an impasse, and the problem solver is called to repair it.

Several bugs can be generated by repairing the impasse:

1. The "no-op" repair heuristic (which says to skip the operation whose precondition is violated) leads to the bug "Blank-Instead-of-Borrow" which simply does not write an answer in the LFS column;
2. The "quit" repair causes the subtraction process to halt at the first LFS column;
3. The "swap vertically" repair generates the interesting "Smaller-From-Larger" bug which results in the absolute difference entered as the answer in LFS columns;
4. The "dememorize" bug of answering 0 in all LFS columns (i.e., $m - n = 0$ when $n>m$) is so called because it involves the inverse of "memorizing" which involves learning of the fact table and semantics of subtraction.

VanLehn points out that not all deletions lead to impasses; some repairs generate bugs which have never been observed in student subjects; and that a repair to one impasse can lead to a second. All of these are issues involving control. For instance, the second, over-generation of bugs, means that the repair proposed must be constrained. VanLehn also discusses the role of "critics" in a generate-and-test architecture of the repair generator to filter down the proposed repairs. In connection with repairs, he offers a taxonomy of repair heuristics.

The Repair Theory of vanLehn and Brown is a good example of "principled" modelling of human mathematical behavior. Once vanLehn and Brown select their approach of repairs, they stick to it (and do not introduce ad hoc repairs to it) to see how far it can be pushed and how it squares with the observed data. It provides a detailed accound of mathematics in a narrow area by an account which is most likely transferrable to other tasks. Of course, using their approach in other tasks would involve background work on, for instance, the domain-specific procedures, impasses, and repairs in the new area.

4. Rissland's work on Constrained Example Generation

Almost all episodes of learning -- whether by person or machine -- depend on having a rich store of examples (worked and posed problems, specific cases, etc.); see for instance the discussion of Mitchell's, Selfridge's and Lenat's work in Part III. Without the experience gained by consideration of examples, learning, understanding, skill acquisition, proving theorems, debugging programs, etc. all come to a halt (or cannot even begin). Despite the central role played by examples, they are often overlooked or taken for granted. Yet, any expert teacher knows not only the value of examples but also that not all examples serve equally well to make a point or provide a test case and what often distinguishes one example from another is the properties it possesses.

Thus with regards to learning and teaching, one does not pick examples at random: they are generated for a purpose -- like giving evidence for or against a conjecture -- and thus are usually (carefully) chosen to possess certain desired properties or _constraints_.

Rissland calls this process of generating examples that meet prescribed constraints "Constrained Example Generation" or "CEG" [Rissland 1980, 1981; Rissland and Soloway 1980]. The CEG model is based upon observations of humans working problems in which they are asked to generate examples satisfying certain constraints. It incorporates three major phases: RETRIEVAL, MODIFICATION, and CONSTRUCTION.

When an example is sought, one can search through one's storehouse of examples for one that matches the properties desired. If one is found, the example generation problem has been solved through RETRIEVAL. In retrieval, there are many semantic and contextual factors -- like the last generated example -- and therefore one is not merely plunging one's hand into an unorganized knowledge base. Thus even though retrieval sounds simple, it can be very complex.

However, when a match is not found, how does one proceed? In many cases, one tries to MODIFY an existing example that is judged to be close to the desired example, or to have the potential for being modified to meet the constraints. Often the order of examples selected for modification is based on judgements of closeness between properties of known examples and the desiderata, that is, how "near" the examples are to what is sought.

If attempts at generation through modification fail, experienced example generators, like teachers or researchers, do not give up; rather they switch to another mode of example generation, which we call CONSTRUCTION. Under construction, we include processes such as combining two simple examples to form a more complex one and instantiation of general model examples or templates to create an instance. Construction is usually more difficult than either retrieval or modification.

<u>General Skeleton of the CEG Model</u>

CEG has subprocesses for: Retrieval, Modification, Construction, Judgement, Control

Presented with a task of generating an example that meets specified constraints, one:

1. SEARCHES for and (possibly) RETRIEVES examples JUDGED to satisfy the constraints from an EXAMPLES KNOWLEDGE BASE (EKB); or

2. MODIFIES existing examples JUDGED to be close to, or having the potential for, fulfilling the constraints with domain-specific MODIFICATION OPERATORS; or

3. CONSTRUCTS an example from domain-specific knowledge, such as definitions, general model examples, principles and more elementary examples.

In examining human protocols, one sees two types of generation: (1) retrieval plus modification; and (2) construction. That is, one does not necessarily try first retrieval, then modification, then construction; sometimes construction is attempted straightaway. Clearly, this model needs many other features to describe the CEG process in its entirety; more details can be found in ιRissland 1981].

To give the reader an idea of the richness and complexity of the CEG process, we present here a synopsis of a CEG problem taken from the domain of elementary function theory. The problem is:

> Give an example of a continuous, non-negative function, defined on all the real numbers such that it has the value 1000 at the point x=1 and that the area under its curve is less than 1/1000.

Most protocols for this question began with the subject selecting a function (usually, a familiar reference example function) and then modifying it to bring in into agreement with the specifications of the problem. There were several clusters of responses according to the initial function selected and the stream of the modifications pursued. A typical protocol went as follows [Rissland 1980]:

> "Start with the function for a "normal distribution". Move it to the right so that it is centered over x=1. Now make it "skinny" by squeezing in the sides and stretching the top so that it hits the point (1, 1000)."
>
> "I can make the area as small as I please by squeezing in the sides and feathering off the sides. But to demonstrate that the area is indeed less than 1/1000, I'll have to do an integration, which is going to be a bother."
>
> "Hmmm. My candidate function is smoother than it need be: the problem asked only for continuity and not differentiability. So let me relax my example to be a "hat" function because I know how to find the areas of triangles. That is, make my function be a function with apex at (1, 1000) and with steeply sloping sides down to the x-axis a little bit on either side of of x=1, and 0 outside to the right and left. (This is OK, because you only asked for non-negative.) Again by squeezing, I can make the area under the function (i.e., the triangle's area) be as small as I please, and I'm done."

Notice the important use of such modification operations as "squeezing", "stretching" and "feathering", which are usually not included in the mathematical kit-bag since they lack formality, and descriptors such as "hat" and "apex".

Another thing observed in <u>all</u> the protocols is that subjects make implicit assumptions -- i.e., impose additional constraints -- about the symmetry of the function (i.e., about the line x=1) and its maximum (i.e., occurring at x=1 and being equal to 1000). There are no specifications about either of these properties in the problem statement. These are the sort of tacit assumptions that Lakatos [1976] talks about; teasing them out is important to studying both mathematics and cognition.

III. A.I. PROGRAMS TO DO MATHEMATICS SOMEWHAT LIKE A MATHEMATICIAN

1. Slagle's Symbolic Automatic Integrator (SAINT)

One of the classic programs in A.I. is Slagle's program written in 1961 as a doctoral dissertation to perform indefinite integration. SAINT (Symbolic Automatic INTegrator) performed at about the level of a good freshman calculus student [Slagle 1962]. A successor to SAINT was Joel Moses' SIN program which has evolved into the highly successful MACSYMA system which can perform exceedingly complicated symbolic mathematics and is used by the professional mathematics commmunity.

The integration problems that SAINT could handle involve a class of elementary functions, defined as followed:

1. any constant is an elementary function
2. the variable is an elementary function
3. the sum of elementary functions is elementary;
4. any elementary function raised to an elementary function as a power;
5. a trigonometric function of an elementary function
6. a logarithmic or inverse trigonometric function of an elementary function;

SAINT works as follows when given a problem:

1. it determines whether the integrand is one of 26 standard forms, that is, when it is a substitution instance of a standard form like INT 2**xdx is an instance of INT c**xdx. If so the answer can be given immediately;
2. if not of standard form, it is tested to see if it is amenable to a well-defined transformation which, when applicable, is usually appropriate, such as factoring the constant outside the integral, interchaning finite summation and integration, making a linear subsitution. There are eight transformation rules.
3. if not of standard form, the integral is tested to see if it is amenable to a heuristic transformation which is a transformation that usually helps with the solution. There are ten heuristic transformations, the most useful of which is the "derivative-divide" rule described below.

To give the reader an idea of how SAINT works, we present the example from [Slagle 1962]:

As a concrete example we sketch how SAINT solved

$$\int \frac{x^4}{(1-x^2)^{5/2}}\,dx$$

in eleven minutes. SAINT's only guess at a first step is to try substitution: $y = \arcsin x$, which transforms the original problem into

$$\int \frac{\sin^4 y}{\cos^4 y}\,dy$$

For the second step SAINT makes three alternative guesses:

A. By trigonometric identities $\int \frac{\sin^4 y}{\cos^4 y}\,dy = \int \tan^4 y\,dy$

B. By trigonometric identities $\int \frac{\sin^4 y}{\cos^4 y}\,dy = \int \cot^{-4} y\,dy$

C. By substituting $z = \tan(y/z)$ $\int \frac{\sin^4 y}{\cos^4 y}\,dy = \int 32 \frac{z^4}{(1+z^2)(1-z^2)^4}\,dz$

SAINT immediately brings the 32 outside of the integral.

After judging that (*A*) is the easiest of these three problems SAINT guesses the substitution $z = \tan y$, which yields

$$\int \tan^4 y\,dy = \int \frac{z^4}{1+z^2}\,dz$$

SAINT immediately transforms this into

$$\int \left(-1 + z^2 + \frac{1}{1+z^2}\right) dz = -z + \frac{z^3}{3} + \int \frac{dz}{1+z^2}$$

Judging incorrectly that (*B*) is easier than

$$\int \frac{dz}{1+z^2}$$

SAINT temporarily abandons the latter and goes off on the following tangent. By substituting $z = \cot y$,

$$\int \cot^{-4} y\,dy = \int -\frac{dz}{z^4(1+z^2)} = -\int \frac{dz}{z^4(1+z^2)}$$

Now SAINT judges that

$$\int \frac{dz}{1+z^2}$$

is easy and guesses the substitution, $w = \arctan z$ which yields $\int dw$. Immediately SAINT integrates this, substitutes back and solves the original problem.

$$\int \frac{x^4}{(1-x^2)^{5/2}}\,dx = \arcsin x + \tfrac{1}{3}\tan^3 \arcsin x - \tan \arcsin x$$

One of the noteworthy mechanisms in SAINT was the use of an "and-or" goal tree. Each integration problem can be treated as a goal which involves the solution of other integration, i.e., subgoal, problems. When there are alternatives, each of which would suffice, to achieving a goal, this represents an "OR" branch in the tree. When there are necessary subgoals, each of which is needed, this represents an "AND" branch.

By use of a "goal stack", Slagle maintains the "focus of attention" in his problem solver. Each time there is a new task to be done, it is added to the goal stack. For instance, an "algorithmlike" or heuristic transformation usually adds goals to the goal stack. Satisfying goals allows the stack to be reduced. Slagle maintains two goal stacks, one of which is the "heuristic goal list" of goals which are neither of standard form nor amenable to standard algorithmlike transformations. This is like keeping a list of things to do.

Part of the determination of the amenability of a goal to a transformation is a list of characteristics which include function type (like rational, algebraic, etc.), and a measure of the depth of the deepest functional composition in the integrand.

The most successful heuristic in Slagle's program was "derivative-divide" which searches for a subexpression s(x) in the integrand g(x) such that s'(x) divides g(x) and results in an expression with fewer factors and then makes the substitution u=s(x) in the integration. For instance, in x INT exp(x**2) dx substitute x**2. SAINT actually discovered the substitution u=exp(x**2).

There are several points to be made about SAINT which are relevant to the mathematics education community:

1. the skill involved in a task like integration can be explicated in procedural terms;
2. much skill knowledge can be described in terms of rules of an if-then nature;
3. Slagles program illustrated the use of goals, heuristics, and knowledge representation.

SAINT was tested on a total of 86 problems, 54 of them chosen from MIT final examinations in freshman calculus, and correctly solved all but two. The most difficult were:

$$\int \frac{\sec^2 t}{1 + \sec^2 t - 3\tan t} \, dt \quad \text{and} \quad \int \frac{x^4}{(1 - x^2)^{5/2}} \, dx$$

2. Lenat's AM program

Lenat's doctoral dissertation program AM (Automated Mathematician) is a knowledge-based program which discovered mathematical concepts [Lenat 1977, Lenat and Davis 1982]. Provided with a rich arsenal of heuristics concerning discovery and interestingness, and a basis of concepts in set theory, it developed concepts in elementary number and set theory, like "prime". Lenat's program has the themes of strong knowledge-based programming, heuristics encoded as if-then rules and an agenda mechanism to control the focus of attention on things to do.

AM is a program which expands a knowledge base of mathematical concepts. Each concept is stored as a frame data-structure. Creating a new concept frame is the principle task for AM; this activity involves setting up a new data structure for the concept and filling in the slots. Filling in a slot is accomplished by executing a collection of relevant heuristic rules. The possible things for AM to do -- like setting up a new concept and filling in a concept slot -- are kept track of by the A.I. mechanism known as an "agenda". AM looks at the agenda of things to do and selects the next task on the basis of considerations such as its importance and interestingness. Filling in a slot, like "Fill-in examples of primes" is an example of a typical task. A heuristic rule is relevant if executing it brings AM closer to satisfying the task. Relevance is determined a priori by pre-determined connections between the heuristic and the slot it effects.

Once a task is chosen from the agenda, AM gathers relevant heuristic rules and executes them. Then AM picks a new task. There are three kinds of effects from such execution:

1. slots of concepts get filled in, for instance a heuristic relevant to filling in an example slot is:
   To fill in examples of X, where X is a kind of Y. Check examples of Y; some of them might be examples of X as well. For instance, to fill in an example for the "prime number" concept, AM would consider examples of the "number" concept.

2. new concepts are created, for instance a heuristic relevant to this task is:
   If some (but not most) examples of X are also examples of Y,
   Then create a new concept defined as the intersection of the concepts X and Y.

3. new tasks are added to the agenda, for instance by execution of the heuristic:
   If very few examples of X are found,
   Then add the following task to the agenda: "Generalize the concept X".

The knowledge of mathematical concepts is encoded in a network of about 115 frames; each frame has several slots describing different aspects of the concept like its NAME, DEFINITION, EXAMPLES, GENERALIZATIONS, SPECIALIZATIONS, and WORTH. AM uses about 250 heuristics. A heuristic (which turned out to be quite powerful) is that dictating the examination of extreme cases.

AM has roughly 40 heuristics that deal with the creation of new concepts. Some are general and apply to any concept; others apply only to functions and relations. The general heuristics include: (1) generalization through such mechanisms as dropping a condition or changing a constant to variable; (2) specialization; (3) exception handling, for instance, if a concept has a negative examples, AM might create a new concept whose instances are the negative examples. Methods that apply to functions and relations include generalization, specialization and inversion.

To give the reader an idea of how AM works, the following is an excerpt from Lenat's thesis [Lenat and Davis 1982, Chapter 2] describes the run in which AM discovered the concept of "prime". (NB. AM does not have any natural language fluency; what follows is a transcript that was edited by Lenat to improve its readability.)

```
**Task 65:**
Filling in examples of the following concept: "Divisors-of".

   3 Reasons: (1) No known examples for Divisors-of so far.
              (2) TIMES, which is related to Divisors-of, is
              now very interesting.
              (3) Focus of attention: A<M recently defined Divisors-of.

26 examples found, in 9.2 seconds. e.g.: Divisors-of(6)={1 2 3 6}
```

```
**Task 66:**
Considering numbers which have very small set of Divisors-of
(i.e., very few divisors).

  2 Reasons:  (1) Worthwhile to look for extreme cases.
              (2) Focus of attention: AM recently worked on Divsors-of.

Filling in examples of numbers with 0 divisors.
  0 examples found, in 4.0 seconds.
  Conjecture: no numbers have 0 divisors.

Filling in examples of numbers with 1 divisors.
  1 example found, in 4.0 seconds, e.g.: Divisors-of(1)={1}
  Conjecture: 1 is the only number with 1 divisor.

Filling in examples of numbers with 2 divisors.
  24 examples found, in 4.0 seconds, e.g.: Divisors-of(13)={1 13}
  No obvious conjecture. This kind of number merits more study.
  Creating a new concept: "Numbers-with-2-divisors".

Filling in examples of numbers with 3 divisors.
  11 examples found, in 4.0 seconds, e.g.: Divisors-of(49) = {1 7 49}
  All numbers with 3 divisors are also Squares. This kind of number merits more study.
  Creating a new concept: "Numbers-with-3-divisors"


**Task 67:**
 ....
Considering the square-root of Numbers-with-3-divisors.
```

This session was preceeded by AM's discovery of multiplication and division; this lead to the concept of "divisors of a number". AM found examples of this concept, in particular AM investigated exteme cases, that is numbers with very few or very many divisors. AM then was ready to discover "primes". As Lenat states, "AM thus discovers Primes in a curious way. Numbers with 0 or 1 divisor are essentially nonexistent, so they're not found to be interesting. AM notices that numbers with 3 divisors always seem to be squares of numbers with 2 divisors (primes). This raises the interestingness of several concepts, including primes." AM then goes on to use the concept of prime in generating various conjectures like "unique factorization" which arise out of the heuristic to ask if the relation between a number and its divisors is a function.

Lenat's AM is an example of a strongly knowledge-based A.I. program which can do quite interesting tasks -- the sort of tasks that if a person did them, one would say that there was intelligence involved. It is a good example of an A.I. program which starts with knowledge and power and is able to generate new knowledge. Thus, AM's strengths are as an A.I. program; Lenat's remarks about its psychological validity are less compelling. However, his program is a very good demonstration of the power of heuristics, knowledge representation and agenda mechanisms.

3. Selfridge's COUNT Program

Selfridge's COUNT program was designed to explore the issue of learning through a teacher challenging a student to solve problems. This work is at the opposite end of the spectrum from Lenat's: whereas Lenat uses a strong knowledge-based approach rich with control and search heuristics, Selfridge's program starts with minimal knowledge and has a very simple control structure.

The task for the COUNT program is to count the number of characters in a string. The program initially possesses a few "primitive capabilities" from which it is to derive a capability, i.e., learn how, to count. COUNT's problem is to learn how to count the number of letters in a string. It has available to it a register ("N"), the position of a pointer ("PTR") and the length of the string ("LENGTH"). Its primitive capabilities affect those three data. The capabilities are:

```
INCREMENT     DECREMENT   EXCHANGE    LEFT    RIGHT
REPEAT        N TIMES
```

INCREMENT and DECREMENT affect the register N; EXCHANGE interchanges two letters in the string; LEFT and RIGHT affect the pointer; REPEAT causes a capability to be repeated until it cannot be performed. N TIMES causes a capability to be repeated N times, where N is the value in the register.

The teacher starts out by being shown a "world state" which consists of a string of letters, a pointer position and a value in N. The teacher then describes a new world state which it wishes COUNT to achieve (this is the "posed problem"). For instance, the teacher might ask COUNT to get the register N to be zero. By a sequence of posed problems and COUNT's discovering how to do them, COUNT can build up new capabilities (which are sequences of prior capabilities) and can eventually, if taught, count the number of letters in the string. Actually the counting problem is just a pre-problem to learning how to reverse the letter string -- a much harder problem.

One important point of COUNT is that the teacher gets COUNT to do more not by programming it but by challenging COUNT to discover new capabilities. This is much the situation with children and other learners: one cannot "make" the learner do the skill, but rather must challenge the learner to develop the needed skills. (Think of getting a child to pitch -- first, the child must learn to throw; or a horse and rider to jump a four foot fence...)

The sequence of posed problems must challenge but not overwhelm the program (the same would be true of children and horses). Overwhelming in the case of COUNT means exceeding its search resources. The program uses "blind breadth-first" search to discover a sequence of actions that solve a possed problem: first search all the capabilities on the current menu; then search all pairs; then all triplets, etc. The current menu consists of the primitive capabilities plus any new ones that the teacher has caused the program to add. This is done by asking the program to save what it has done in the successful solution of a posed problem (it is given a name of the teacher's choosing). Thus, new capabilities only get added in response to success on a posed task. COUNT provides a demonstration that a program can learn without highly sophisticated control techniques, but the subtlety comes in the selection of problems to pose.

The following is an excerpt of a session to teach COUNT how to count. COUNT presents a tableau of its world state and the teacher responds by posing a new world state:

```
  A B C D E                     N=16
        ^                       (Ptr= 4   Length=  5)

TYPE PROBLEM:
  *
  0
  *
```

The order of specifying the posed world state is: (1) the new string; (2) the new value in N; (3) the new pointer position. An asterisk indicates the teacher doesn't care about the value. In this first problem, the teacher is solely interested in getting the register to be zero, N=0 and doesn't care about the actual letters in the string nor the position of the pointer.

COUNT responds to this first challenge:

```
    I think I've got it.
    Yes, it works. Shall I try it again? Y or N: N.

    Here's new status. Pose problem again. If you don't like
    status, Type S (and return).

      A B C D E F G H I          N=6
          ^                      (Ptr= 3  Length= 10)
```

The most streamlined way to teach COUNT to count is:

1. zero out the register N (call it "NZERO")
2. move the pointer all the way to the left and zero out the register ("INIT")
3. move the pointer one letter right and add one to N ("MOVE-RIGHT-AND-ADD")
4. count the number of letters in the string

Thus at the end of teaching COUNT to count, four new capabilities (NZERO, INIT, MRA, COUNT) have been added to COUNT's repertoire of capabilties. Note, most people end up with more capaibilties on the menu simply because they too are performing a "search" of problems to pose and solutions for COUNT to remember. It is possible to impede COUNT by asking it to remember too many capabilities because this causes an increase in the size of the space COUNT must search for answers to new problems.

Selfridge's program provides a laboratory to study issues in learning like the role of the teacher, posed problems, search. He does not claim the humans necessarily learn like COUNT (with emphasis on search). He does claim that humans do need exercise and evaluation of their capabilities to learn. One could claim that early learning -- when one is like an infant with regards to a task -- might very well be something like COUNT for what else can one do but try with what capabilities one possesses. The other end of the learning spectrum is a program like Lenat's where the learner has a vast, rich, well-structured body of knowledge to bring to bear on new problems. Note that regardless of the kind of learning, the examples, data and experiences involved are critical. I would claim that without a rich body of examples there can be no learning.

4. Mitchell's LEX program

Tom Mitchell has built a program LEX to Learn by EXperimentation in the domain of symbolic integration, the same domain as Slagle's much earlier work [Mitchell 1983]. Where Slagle's SAINT was endowed with certain heuristics, like "derivative-divide", which it used in its task of doing integration problems, the task of Mitchell's LEX is to learn such heuristics. In other words, LEX's task is to learn the kind of knowledge that made SAINT powerful.

Mitchell's work on LEX is an application of his idea of "version spaces" which he developed in his doctoral thesis [Mitchell 1978]. Version spaces are a mechanism to represent the range of incompletely learned heuristics in terms of the possibilities spanned by the most specific case and the most general case of successful application of the heuristic.

Mitchell is attacking a central and classic problem in A.I., namely "learning". It is relevant to the concerns of this paper because some of its methods can be transferred to the teaching of mathematics; in particular the notion of refining one's notion of a heuristic on the basis of "positive" and "negative" examples. LEX thus forms a bridge between the concerns of Schoenfeld and those of Slagle, Lenat and the A.I. learning community. It also is another case in point on the ubiquitous role of examples.

LEX acquires and modifies heuristics by iteratively cycling through the processes of: (1) generating a practice problem; (2) using available heuristics and other knowledge to solve this problem; (3) analyzing and critizing the search steps in obtaining the solution; and (4) proposing and refining new domain-specific heuristics. LEX, like other A.I.

programs, starts out with some initial store of domain-specific knowledge as well as an architecture embodying general knowledge about learning, representation, control, etc. Its initial domain-specific knowledge consists primarily of two sorts, much like SAINT:

1. operators -- these include heuristic "algorithmlike" (to use Slagle's term) transformations as well as "book knowledge" like common anti-derivatives and standard transformations. The operators are stored in "If-Then" format, where the "If" clause contains the conditions necessary for application of the operator, and the "Then" contains the result of the application. For example:

    1. OP1: INT r.f(x)dx ===> r .INT f(x)dx
    2. OP2 (Integration By Parts):
       INT u dv ===> uv - INT v du
       (This is represented as INT f1(x)f2(x)dx, where f1(x) corresponds to u and f2(x)dx corresponds to dv.)
    3. OP3 1.f(x) ===> f(x)
    4. OP4 INT f1(x)+f2(x) dx ===> INT f1(x)dx + INT f2(x)dx
    5. OP5 INT sin(x) dx ===> -cos(x) + C
    6. OP6 INT cos(x) dx ===> sin(x) + C

2. a type heirarchy -- which lays out the relationships between major types of domain items, like functions, which LEX must manipulate. LEX derives much of its power to learn from this heirarchy which essentially captures the notion of generality in LEX's domain. For instance the specific functions sin, cos and tan are trig functions and ln and exp are expln functions; trig and expln functions are transcendental functions. The identity function, constant function, integer exponent are monomials which in turn are polynomials.

LEX's task is to learn when a heuristic should be applied and how. For instance, in the case of Integration by Parts ("IBP"), LEX is to learn how to "bind" the u and the dv. As any who has learned or taught calculus knows there is some art in choosing the u and the dv; LEX is trying to discover and express that art.

The LEX program contains four modules: Problem Solver, Critic, Generalizer, and Problem Generator. Their functions are as follows:

1. Problem Solver tries to solve the problem at hand with its available store of operators, including the current status of its heuristics;
2. Critic analyzes the trace of a successful solution to glean positive and negative instances. A positive instance is a problem state on the way to a successful solution; a negative instance is on a path that led away from the solution;

3. Generalizer rewrites its knowledge of heuristics on the basis of what the Critic tells it: it squeezes in from the most general statement of the heuristic on the basis of negative instances and pushes out from the most specific on the basis of postive instances;
4. Problem Solver poses new problems to solve which will help to further refine knowledge of the heuristics.

For instance, suppose LEX is trying to learn IBP (that is, refine OP2 to narrower classes for f1(x) and f2(x)) and has been posed the problem:

INT 3x sinx dx

At the completion of one cycle, IBP has been refined and is narrowed to a range of possibilities from:

"most specific" : Apply IBP with u=3x and dv=sin(x)dx

"most general": original form of OP2 (with f1(x) and f2(x))

This range is represented by a version space that captures all the intermediate possibilities like:

Apply IBP to INT 3x trig(x) dx with u=3x and dv=trig(x)dx
Apply IBP to INT poly(x) cos(x) dx with u=poly(x) and dv=cos(x)dx
Apply IBP to INT kx trig(x) dx with u=kx and dv=trig(x)dx (k an integer)

Eventually LEX homes in on an intermediate case.

One lesson for mathematics education from this work is to mimic the learning cycle of LEX. That is, make it an explicit task fir students to learn "what should be the u and what should be the dv" and discover this by trying problems -- albeit wisely chosen -- of positive and negative force to narrow down on the answer. Another more implicit message is that domain-knowledge such as LEX's heirarchy of types is exceedingly powerful and should be made explicit to students; this is a point made by Rissland.

Note that LEX still isn't clever enough to know to group INT (sinx)**2 + INT f(x)dx + INT (cosx)**2 to take advantage of the obvious identity and arrive at X + INT f(x)dx. Because it does not have concepts like "even integer" and "odd integer", it can't learn some of the usual tricks involving integrals of powers of sin and cos (which involve one trick when there exists an odd power and a different one for both even powers).

An example of one cycle of LEX starting with the problem INT 3x cos(x)dx results in the flow of information and a version space summarized by the following figure (Fig. 6-5 from his paper in [Michalski, Carbonell and Mitchell 1983]):

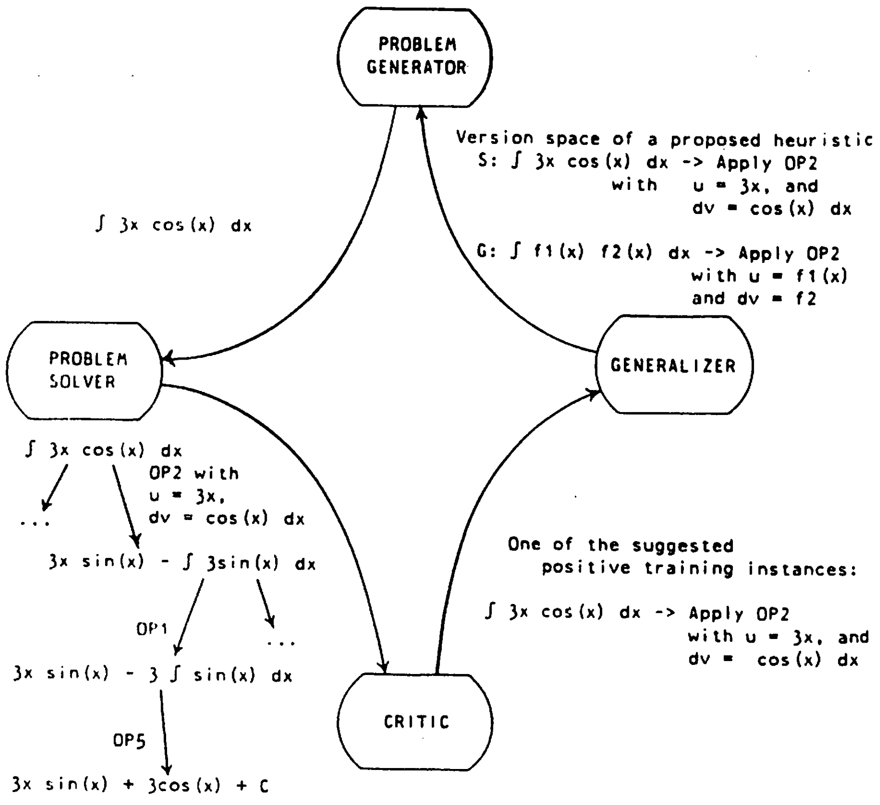

PROBLEM GENERATOR
Version space of a proposed heuristic
S: ∫ 3x cos(x) dx -> Apply OP2 with u = 3x, and dv = cos(x) dx
G: ∫ f1(x) f2(x) dx -> Apply OP2 with u = f1(x) and dv = f2
∫ 3x cos(x) dx
PROBLEM SOLVER
GENERALIZER
∫ 3x cos(x) dx
...
OP2 with u = 3x, dv = cos(x) dx
3x sin(x) - ∫ 3sin(x) dx
OP1
...
3x sin(x) - 3 ∫ sin(x) dx
OP5
3x sin(x) + 3cos(x) + C
One of the suggested positive training instances:
∫ 3x cos(x) dx -> Apply OP2 with u = 3x, and dv = cos(x) dx
CRITIC

Figure 6-5: The learning cycle in LEX.

IV. CONCLUSIONS

My purpose in presenting the above dozen samples of research was not only to give the reader an idea of what A.I.-style research is and pointers into the relevant literature but also to show, by example, that many questions central to A.I. research are central to mathematics education. In particular, both fields are interested in questions such as:

1. How do we learn?
2. What is understanding?
3. What is the knowledge involved in mathematical expertise?
4. How can we describe and represent such knowledge?
5. How can we impart such knowledge to students?

I believe that the tools of A.I., in particular those of knowledge representation, process description, and planning/control, provide a useful kitbag for the mathematics educator in grappling with such questions. A.I. concepts make it easier to describe what we know and intuit about learning and, if we want, to experiment and test out our ideas by implementing them as programs. This expressive power makes it possible to de-mystify and describe our knowledge: how it is structured, stored, acquired and refined. Applying this approach to expert mathematical problem-solving is especially relevant to mathematics education since by better understanding and describing such knowledge, we should be better able to transfer it to our students and thereby help them improve.

Another "epistemological" lesson A.I. teaches is that it is very important to attend to "meta" and "tacit" knowledge: for example, strategies for understanding and non-formal aspects of mathematical knowledge. A.I. programs have demonstrated that such knowledge is central to machine intelligence. This can be taken as a strong hint that it might also be vital to human intelligence.

As mathematicians and educators, we may not be satisfied with the descriptions and programs of A.I. but they do give us a place to begin, even if only in reaction and criticism. A.I. offers existence proofs that important mathematical skills can be understood and detailed. A.I. programs, while they might not prove the necessity of certain ingredients of knowledge, can demonstrate sufficiency. While A.I. programs do *not* prove psychological validity for such detailed models, they can be used to empirically evaluate and test the sensitivity of models. And they provide a rigorous medium for testing out ideas.

A.I. thus offers the mathematics education researcher a body of concepts with high expressive power, demonstrations of the value of "meta" and tacit knowledge, a rigorous medium for testing out ideas about learning and some strong hints about what is important for learning in humans. A.I. has taught us not to be afraid to tackle hard questions such as what constitutes expertise. We should take courage from this. In such research, the important point is not to be exactly right the first time, but rather to begin and then to evolve. I believe the same is true, by the way, for (A.I.) programs. A.I. offers us a way to begin.

References

Barr, A., and Feigenbaum, E. A., (Eds.) The Handbook of Artificial Intelligence. Volumes 1 and 2, Wm. Kaufman, Inc., Los Altos, CA, 1981.

Borning, A., and Bundy, A., "Using Matching in Algebraic Equation Solving". Proceedings International Joint Conference on Artificial Intelligence (IJCAI-81). Vancouver, B. C., 1981.

Brown, J. S. and Burton, R., "Diagnositc Models for Procedural Bugs in Basic Mathematical Skills", Cognitive Science, Vol.2, 155-192, 1978.

Brown, J. S., and vanLehn, K., "Repair Theory: A Generative Theory of Bugs in Procedural Skills", Cognitive Science, Vol. 4, 379-426, 1980.

Bundy, A., Analysing Mathematical Proofs (or reading between the lines). Research Report DAI No. 2, Department of Artificial Intelligence, University of Edinburgh, 1975.

Bundy, A., and Silver, B., "Homogenization: Preparing Equations for Change of Unknown". Proceedings International Joint Conference on Artificial Intelligence (IJCAI-81), Vancouver, B. C., 1981.

Burton, R.R., "Diagnosing bugs in simple procedural skills", Intelligent Tutoring Systems. In Sleeman and Brown (Eds.) Intelligent Tutoring Systems, 1982.

Cohen, P. C., and Fegeinbaum, E. A. (Eds.) The Handbook of Artificial Intelligence. Volume 3, Wm. Kaufman Inc., Los Altos, CA, 1982.

Davis, R. B., Young, S., McLoughlin, P., The Roles of "Understanding" in the Learning of Mathematics Curriculum Laboratory, University of Illois, Urbana/Champaign, April 1982.

Davis, R. B., Jackson, E., and McKnight, C., "Cognitive Processes in learning algebra." Journal of Children's Mathematical Behavior, 2, No. 1, 1978.

Davis, R. B., and McKnight, C., "The Influence of Semantic Content on Algorithmic Behavior ". Journal of Children's Mathematical Behavior, Vol. 3, No. 1, 1980.

Feigenbaum, E. A., and Feldman, J., (Eds.) Computers and Thought. McGraw-Hill, New York, 1962.

Lakatos, I., Proofs and Refutations. Cambridge University Press, 1976.

Lenat, D. B., Automated Theory Formation in Mathematics, Proceedings International Joint Conference on Artificial Intelligence, MIT, Cambridge, Mass, 1977. Available from Wm. Kaufman, Inc., Los Altos, CA.

Lenat, D. B., and Davis, R., _Knowledge-Based Systems in Artificial Intelligence_. McGraw-Hill, New York, 1982.

Matz, M., "Towards a process model for high school algebra errors". In Sleeman and Brown (Eds.) _Intelligent Tutoring Systems_, 1982.

Matz, M., "Towards a Computational theory of Algebraic Competence", _Journal of Children's Mathematical Behavior_, Vol. 3, No. 1, 1980.

Michalski, R. S., Carbonell, J., and Mitchell, T., (Eds.) _Machine Learning_. Tioga Publishing Company, CA, 1983.

Mitchell, T. M., "Learning and Problem Solving". _Proceedings International Joint Conference on Artificial Intelligence (IJCAI-83)_, Karlsruhe, West Germany, 1983. Available from Wm. Kaufman, Inc., Lost Altos, CA.

Mitchell, T. M., _Version Spaces: An approach to concept learning_. Ph. D. thesis, Stanford University, December 1978. Also report STAN-CS-78-711, Stanford University.

Papert, S., _Mindstorms_. Basic Books, New York, 1980.

Polya, G., _How To Solve It_. Second Edition, Princeton University Press, 1973.

Rissland, E. L., _Constrained Example Generation_, Technical Report 81-24, University of Massachusetts, Amherst, MA, 1981.

Rissland, E. L., "Example Generation", _Proceedings Third National Conference of the Canadian Society for Computational Studies of Intelligence_, Victoria, B.C., May 1980.

Rissland, E. L., "The Structure of Knowledge in Complex Domains". In Chipman, Segal and Glaser (Eds.), _Thinking and Learning Skills: Research and Open Questions_, Lawrence Erlbaum Associates, 1984.

Rissland, E. L., _The Structure of Mathematical Knowledge_. Technical Report No. 472, MIT Artificial Intelligence Laboratory, August 1978.

Rissland, E. L., "Understanding Understanding Mathematics", _Cognitive Science_, Vol. 2, No. 4, 1978.

Rissland, E. L., and E. M. Soloway, "Overview of an Example Generation System", _Proceedings First National Conference on Artificial Intelligence_, Stanford, August 1980.

Samuel, A. L., "AI, Where it has been and where it is going". _Proceedings International Joint Conference on Artificial Intelligence (IJCAI-83)_, Karlsruhe, West Germany, 1983. Available from Wm. Kaufman, Inc., Los Altos, CA.

Schank, R. C., "The Current State of AI: One Man's Opinion". _AI Magazine_. Vol. 4, No. 1, 1983.

Schoenfeld, A.H., "Beyond the Purely Cognitive: Belief Systems, Social Cognitions, and Metacognitions as Driving Forces in

Intellectual Performance", Cognitive Science, 7, 329-363, 1983a.

Schoenfeld, A.H., "Theoretical and Pragmatic Issues in the Design of Mathemtical `Problem Solving' Instruction". Paper presented at the 1983 Annual Meeting of the American Educational Research Association, Montreal, Canada, April 1983b.

Schoenfeld, A.H., "Teaching Problem Solving Skills", American Mathematical Monthly, 87, 794-804, 1980.

Schoenfeld, A.H., "Presenting a strategy for indefinite integration", American Mathematical Monthly, 85, pp. 673-671, 1978.

Slagle, J. R., "A Heuristic Program that Solves Symbolic Integration Problems in Freshman Calculus". In Feigenbaum and Fledman (Eds.) Computers and Thought, 1962.

Sleeman, D., and Brown, J. S., Intelligent Tutoring Systems. Academic Press, New York, 1982.