JOURNAL OF MATHEMATICAL BEHAVIOR 5, 305-317 (1986)

Teaching Mathematical Induction I

ED DUBINSKY
Clarkson University

A prototype version of a novel approach to teaching mathematical induction was used in a small class of eight college students. The instructional treatment is based on a Piagetian theory of learning abstract mathematical concepts in which the learner uses reflective abstraction to construct new schemas out of old ones in a hierarchy that ultimately reaches the desired concept. The treatment uses certain computer experiences in an attempt to induce the student to make the appropriate reflective abstractions. The method is seen to be reasonably effective and several areas of possible improvement are indicated.

INTRODUCTION

This paper is, to an extent, a continuation of Dubinsky & Lewin (1986) which was an attempt to formulate the beginning of a theory of learning abstract mathematical concepts at the post-secondary level. The theory was applied to, among other things, mathematical induction and so a major portion of that paper might be entitled, "Learning Mathematical Induction." In the present work we turn to the question of designing and implementing instructional methods for teaching induction that are based on our understanding of how students can learn this concept. Computer experiences will play an important role in these methods.

Mathematical induction is very hard for those undergraduates who, although reasonably successful in other subjects, do not have special talents for mathematics. It seems to be generally agreed by college mathematics teachers that, in general, our students do not come to understand this concept. Indeed, if you question students—even those who have had several mathematics courses—although almost all of them will have heard of induction, not many of them will be able to say anything intelligent about what it is, much less actually use it to solve a problem. Some students won't even be able to recall any kind of problem to which induction can be applied. Those that can are limited in their responses to the problem of proving that a given formula is equal to a certain finite sum. Very few students can actually construct an induction proof and not many can understand one. Refreshing their memory is of little value and does not lead to much success in using induction to prove any statement other than one which is similar to those for which the students have had many illustrations.

Correspondence and requests for reprints should be sent to Ed Dubinsky, Mathematics Department, Clarkson University, Potsdam, NY 13676.

In Dubinsky & Lewin (1986) we tried to explain why. Our theory (relying heavily on the ideas of Piaget) is that learning a concept involves a construction. The learner begins with certain schemas already present and uses the process of reflective abstraction to construct other schemas. By a *schema* here we refer to a collection of mental objects and mental operations which the subject is able to perform on these objects. The description of a construction process for a particular schema is called a *genetic decomposition* of the concept. It describes how the students are observed to learn the concept, that is, acquire the schema.

Applying this to induction, we obtained the genetic decomposition shown in Figure 1. In this chart, the nodes represent schemas and the edges are reflective abstractions that are applied to these schemas to construct new ones (see below for more details). Our claim is that these constructions do not occur spontaneously for most students and teaching must include activities that will stimulate the student to make them. Classroom discussion of induction, however, usually has nothing to do with building these schemas so that most students have no structures with which to assimilate either the explanation or the examples. The result is that the student is reduced to looking for behavior which can be imitated (e.g., on examinations) and little learning takes place.

In Dubinsky & Lewin (1986) we suggested that better results might be obtained if instruction included activities intended to directly stimulate the student to perform the reflective abstractions necessary to construct the required schemas. Explanations could then be based on these schemas and examples could serve to illustrate them and solidify their presence in the minds of students.

In this paper we describe a preliminary prototype effort in this direction. Although it represents no more than a beginning of an implementation, for induction, of the above suggestions, the results are encouraging, both in terms of students' development along the lines of our genetic decomposition and in their success in using induction to prove various statements—including some they have not seen before.

We begin with a more detailed description of the genetic decomposition of induction. This is followed by a very brief mention of some properties of the programming language SETL which was used, and a general discussion of how we used computer experiences to induce learning. Then we describe the course in which induction was taught, the students who took that course, the instructional treatment that was used (including computer experiences), and the means of evaluating what learning took place. Next follows a description of the results of those evaluations and an attempt to interpret them. We conclude with a discussion of what further efforts seem to be called for in this direction.

GENETIC DECOMPOSITION OF INDUCTION

The chart given in Figure 1 was derived in Dubinsky & Lewin (1986) from interviews with students in the the process of learning induction. This was done to illustrate our general theory of constructing cognitive schemas from existing schemas by means of reflective abstraction.

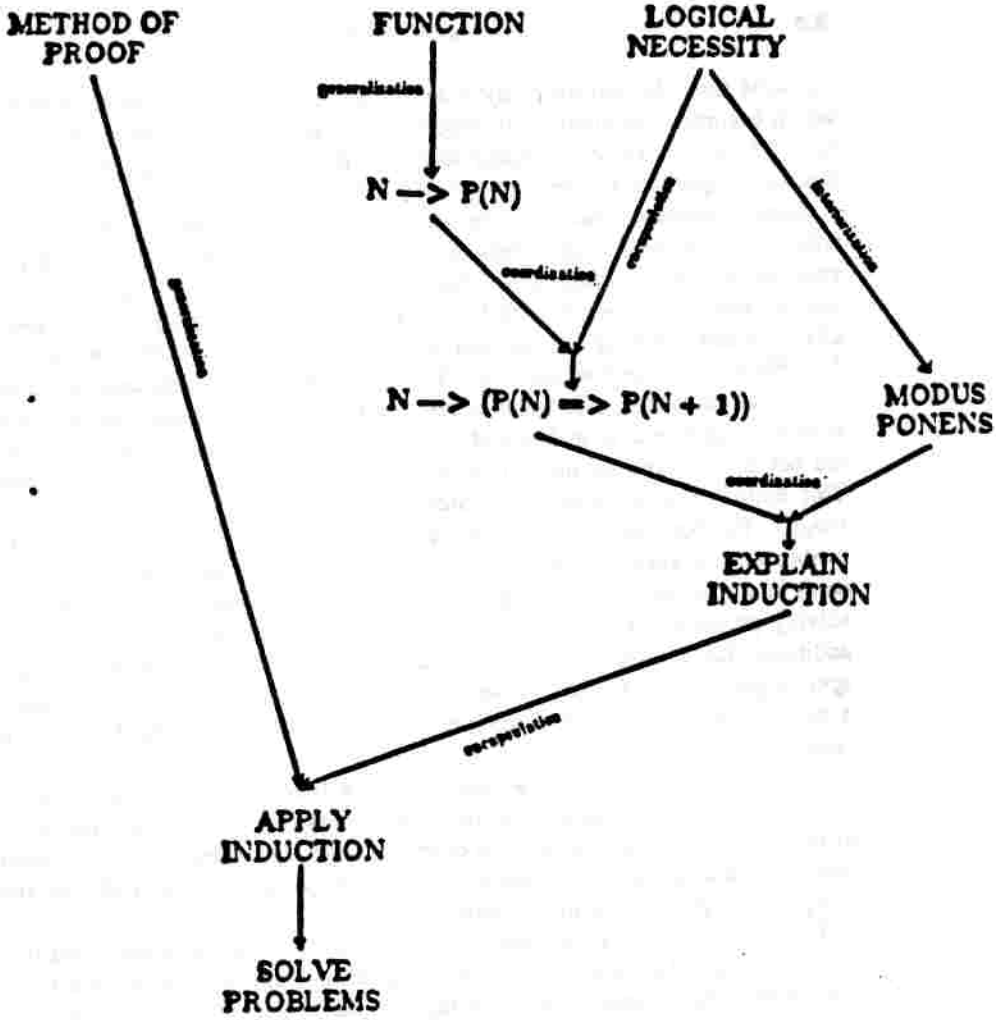

FIG. 1. Genetic decomposition of mathematical induction.

The three schemas at the top of the figure are assumed to be present when the student begins to study induction (although reminders may be necessary). *Method of proof* refers to the students' experience with geometry proofs, proof by contradiction, proof by counterexample and so on. The notion of *function* that is assumed is just an algebraic expression with letters for which numbers may be substituted to obtain a result. There may be a vague notion of domain and range. *Logical necessity* is the very basic concept that, in certain situations, if A is true, then one can be absolutely sure that B is the case.

The first step in the construction is to *generalize* the *function* schema to include a *proposition-valued function* of the positive integers, $N \dashrightarrow (P(N))$. At

the same time, *logical necessity* is *encapsulated* into the implication $A => B$ which becomes, cognitively, an object which can serve as the value of a function. These two are *coordinated* to obtain the schema of *implication-valued function* of positive integers, $N --> (P(N) => P(N + 1))$.

Logical necessity must also be *interiorized* into a process, *modus ponens*, whereby the student is conscious of receiving the inputs A, $A => B$ and then concluding B. *Implication-valued function* is coordinated with *modus ponens* to obtain a process that begins with $P(1)$ (evaluation of P at 1) and then successively alternates evaluation of the *implication-valued function* $N --> (P(N) => P(N + 1))$ with *modus ponens* ad infinitum. The result is the schema, *explain induction*.

It is an interesting side point to observe that there occur many examples of students who appear to understand induction and can discuss it intelligently but are not able to even set up an induction proof—much less actually make one. This seems to occur with other concepts as well and has been reported previously, for example in the problem-solving literature. Schoenfeld (1983) describes several examples in which students apparently understand various concepts in elementary calculus and high school geometry, yet do not use them in solving problems. These situations differ from ours in that they involve the additional issue of knowing when to apply a particular concept in solving a specific problem. In any case, it seems clear that even after a learner understands a concept at the level of being able to explain it, further development is necessary.

For induction, our interviews suggest that a first step is to *encapsulate* the *explain induction* schema from a cognitive process into an object that can serve as an aliment to the schema *method of proof* so that the student is ready to *apply induction;* that is, set up an induction proof properly for most problems and complete the proof for some of them.

Finally, the student must have experience with a multitude of problems that can now be attacked in a general way using this existing schema. Such activity will permit the student to pick up the various specific "tricks" relative to individual situations that are also necessary before he or she is really able to *solve problems*. This last step is the most important but it seems that rarely do undergraduates even reach the point of being able to *explain induction* so that, even though they may be given practice with a large number of examples, lacking a schema with which to assimilate and organize these "experiences," they do not develop any autonomous skills relative to induction.

SETL AND THE USE OF COMPUTER EXPERIENCES

SETL is a very high level procedural programming language with standard constructs of assignments, procedures, for-loops, while-loops, and so on, and the usual simple data types such as integers, floating point, and Boolean along with the usual operations on them. It also has a number of compound data types

such as tuple (array of finite but unbounded, dynamically determined length) and finite set. Components of tuples and elements of sets can be of heterogeneous type including other compound types. Nesting is unlimited.

Type declarations and sizes of compound types need not be specified in a SETL program. These quantities are determined by the context and change automatically as required by the program. There is an explicit value, OM, which means "undefined." Thus a tuple may be considered to be an infinite sequence for which only finitely many terms have been defined.

Because of this flexibility of data specification, fairly complicated SETL programs may be written quickly and easily. An even more helpful feature is the existence of certain mathematical constructs such as map (set of tuples of length 2) and quantification over finite sets. The syntax for these is very similar to standard mathematical notation and, therefore, large portions of SETL programs read like mathematical expressions.

For example, consider the following SETL statement:

```
A := {[x, x**2]: x in {1..1000} | (forall y in {2..x-1} | x mod y /= 0)};
```

It reads, "let A be the set of ordered pairs $[x, x^2]$ such that x is an integer between 1 and 1000 with the property that for each y between 2 and $x-1$, it is the case that y does not divide x." Thus A is the map that assigns, to each prime less than 1000, its square. After the program has executed this statement, the expression $A(7)$, for example, will have the value 49 and $A(8)$ the value OM (undefined).

Imagine how much basic mathematics one must be familiar with in order to understand the content of the previous paragraph. This is the key to our use of computer experiences. Each experience, each computer task, is designed so that understanding the syntax and making it work in various mathematical situations corresponds to one or more of the reflective abstractions required to construct a schema and/or the schema itself. Our working hypothesis is that performing these tasks will induce the student to make the appropriate constructions in her or his mind. This can happen spontaneously or it can be helped along in lectures, discussions, and so on.

For more details on SETL, see Schwartz et al. (1986).

METHOD

The prototype effort we are describing took place in a one semester course given in Spring 1985, at Clarkson University, a small school in upstate New York which concentrates on Engineering, Management, and Science. The class consisted of 8 second-semester sophomores with various majors. All of the students had taken a number of computer science courses as well as several calculus courses. The course in question was the second semester of a year sequence in Finite Mathematics taught with the help of the computer as described above.

In the previous semester, the students had learned to program in SETL and these experiences were used in the above spirit to teach topics such as propositional calculus (including quantification), elementary set theory, relations and functions, vectors and matrices, permutations and combinations, and elementary probability. There was no selection process for the course. It was simply an elective and did not meet any specific requirements. The 8 students (out of 46) who elected to go on to the second semester were not necessarily the best students in the class. Indeed their grade distribution was somewhat below the class average.

Mathematical induction was a major topic for the second semester. Throughout the term an attempt was made to incorporate, into the regular material, activities designed to induce the students to construct the schemas appropriate for induction. At about the middle of the semester, two 75-minute classes were devoted to explicit discussion of this topic. This was the only classroom treatment of induction. We will try to explain how the activities and the discussion were related to the construction processes described above in connection with the genetic decomposition in Figure 1.

Method of proof was mentioned in the lecture. The students were reminded of methods they might be familiar with (high school geometry, proof by contradiction, proof by counterexample, checking all cases, etc.). The only connection with computers was a brief mention of the issue of proving that a program is correct.

A large portion of the semester was devoted to the development of the students' concept of function. They wrote many programs and it was pointed out that a program was a function implemented as a process. Maps in SETL implemented a function as a set of ordered pairs. Specific tasks with evaluation of functions in these different modes, composition, inverses, and using functions to model various situations, were expected to induce the students to adopt a more general notion of function.

In order to make the specific generalization to *proposition-valued function*, students worked with tuples of Boolean values (true or false). They were asked to model situations as tuples, write a program that would convert a procedure or a map to such a tuple, and to use a tuple to evaluate a *proposition-valued function* of integers.

To induce the next generalization—to *implication-valued function*—students were given tasks in which they constructed or were given a tuple of Boolean values, P, and were asked to write a procedure that accepted an integer N and evaluated the expression, $P(N)$ implies $P(N + 1)$.

For *modus ponens*, the students were reminded of if . . . then . . . clauses with which they were quite familiar.

In the two lectures, the students were asked to think about an induction proof as a task of taking a Boolean tuple P whose values were unknown (undefined) and finding a justification for setting each component (from some point on) to the

value "true." Thus, the method of proof by induction was decomposed into the following three sub-tasks.

a) Determine the *proposition-valued function* of integers P that models the particular problem.
b) Determine N_0 and evaluate $P(N_0)$ to see that it is true (i.e., prove that $P(N_0)$ holds).
c) Devise some mechanism that will successively fill in the subsequent values of the components $P(N)$, $N \geq N_0$ (i.e., prove that $P(N) \Rightarrow P(N+1)$ for $N \geq N_0$).

Three examples of induction proofs were worked out in class and they are given in Figure 2. As nearly as could be determined, the students' total experience with induction problems consisted of these examples and previous experience restricted to statements asserting that a certain sum of N terms is equal to a certain arithmetic expression in N. Regarding the latter, their memories seemed quite vague and no student was able to recall a specific problem.

There are certain inadequacies in this preliminary version of our instructional treatment for induction. Not enough was done for *modus ponens* and nothing was done regarding the encapsulation of *logical necessity* as a cognitive object. The treatment of the generalization to *implication-valued function* was weak. The schema *explain induction* probably calls for more than the verbal treatment given here. It would be very nice to have some computer experience that simulates the working of an induction proof. The entire treatment (instruction, practice, and evaluation) involved a total of only six distinct problems which is probably much too small. The sequencing of instruction followed the genetic decomposition fairly strictly. It seems likely, however, that cognitive development in students

Problem 1.
Prove that the following equality holds for all positive integers, N.

$$\sum_{i=1}^{N} i^2 = \frac{N(N+1)(2N+1)}{6}$$

Problem 2.
Prove that the sum of the interior angles of a convex polygon of N sides is $(N-2)\pi$.

Problem 3.
Assuming that any prime which divides the product of two integers must divide one of them, prove that a prime which divides a product of integers must divide one of them.

FIG. 2. Induction problems discussed in class.

Problem 1.

Let (x_n) be a sequence of numbers defined by:

$$x_1 = 1, \quad x_{n+1} = 2^n x_n \text{ for } n = 1,2, \ldots$$

Find a formula for x_n and prove it by mathematical induction.

Problem 2.

Prove by mathematical induction that any sufficiently large number of dollars can be obtained by using only bills of denomination \$5 and \$3 (assuming that such bills exist).

FIG. 3. Induction problems given in take-home assignment.

does not proceed in a totally uni-directional manner, but rather oscillates back and forth between schema as they are constructed—at first tentatively, breaking down when faced with difficulties, and gradually becoming more robust. In addition, examples of induction problems should not be delayed until the student is ready to *explain induction*. They should be presented early and piecemeal so as to help the student construct individual schema. Finally, nothing was done in the course to present problems in a broader context with induction arising naturally as a tool to be applied as part of a larger solution of some problem.

Nevertheless the performance of the students was encouraging. This performance was evaluated in two ways. First, a take-home assignment consisting of two induction proofs was given. The problems are presented in Figure 3. Second, each student was interviewed for 20 minutes on induction. The questions asked are presented in Figure 4. They are the same as those used in Dubinsky &

Question 1.

Tell me, in your own words, what proof by induction means to you. What is involved in making such a proof?

Question 2.

Once you have done this, how can you be sure that the statement you have proved is true, say for n = 7? How would you explain this in every day terms to someone who has no mathematical background?

Question 3.

What mental images do you use in thinking about induction? Suppose you wanted to prove a statement for every negative integer? For every positive integer divisible by 3?

Question 4.

The student is asked to prove by induction that

$$N! > 2^N$$

for N sufficiently large. Paper and pencil is provided.

FIG. 4. Interview questions.

Lewin (1986) and include an induction proof which the student is asked to construct on the spot.

RESULTS

First we consider the assignment (Figure 3). As one might expect, the first problem gave difficulty because the phenomenon of defining a sequence by induction appeared to be completely new for these eight students. Nevertheless, four of them obtained a correct formula and all but one of these set up the induction argument properly and completed the proof correctly.

They did much better with the second problem which was a straight induction proof although apparently quite unlike any problem they had seen before. In this case, seven of them set up the argument properly and five of these completed the proof correctly.

The interviews provided a considerable amount of information. Almost all of the students were able to *explain induction* reasonably. Five of their explanations seemed to contain the essential idea. Here is one example.

> S1: It means that you have something that you want to prove like an equation or something. First you have to show that it will work for some value and then you assume that it will work for some value called k. And then by using your assumption you try to prove that it will work for the next number higher than k, $k + 1$.

Two of the students needed a little prompting regarding the base case. For example:

> S2: . . . method of proving a situation by stating that one condition implies the next condition in a chain of true or false conditions.
> P: Is that enough?
> S2: You can use it for different sorts of problems so I don't know if I want to become too specific.
> P: What I mean is that you said that you prove that one condition implies the next condition. Is that all you have to do?
> S2: And that this holds forever, basically. In other words, one condition always implies the one that comes after it.
> P: That's part of it, there's no question. But there's another part.
> S2: And that you establish a base situation, a first condition

Only one student gave an answer that might be considered unsatisfactory—at least in terms of verbal expression.

> S3: Proving something in a small case so that it can be applied to an infinite series of expressions. You prove a certain expression which therefore is true for a long series of expressions.

All of the students seemed to understand, in a concrete way, that an induction proof provided a mechanism for proving a statement true for each value, either

by beginning with a "base value" for which it could be shown to be true and proceeding step-by-step to the desired value, or by doing the same thing in reverse. Thus, their responses to the second question in Figure 4 were rather good. The best of them was given by S4.

> S4: Well, if its true for 1, and we've shown that if its true for a k, and its then true for a $k + 1$, we'll pretend that k is equal to 1, and by what we've shown, it's true for 2, so we'll try the same thing again, but k will be equal to 2, which means that it's good for 3, and continue up until 7.

The worst of them, given by S3, was not all that bad.

> S3: I'd try to explain that it's a series of associations going all the way back to the original proof, you can show that it's true for one more. Therefore it's true for 2, and I'd try to show the person that through that method, it's going to be true for all the way for any k.

As for the actual problem, all but one student, S3, were successful in solving it, either verbally or on the sheet of paper that we provided for that purpose.

It is possible to see, in these interviews, comments that correspond to our genetic decomposition of induction. In Dubinsky & Lewin (1986) we indicated the process of reflective abstraction by showing examples, for each schema, of a student who was on the verge of constructing it and one who had apparently just done so. The students in the present study (although much less advanced in their college careers), are further along in their development of the concept of induction so there are not many such examples. What we can show, however, are examples (where they are present) of each schema (node on the chart in Figure 1) that must be constructed in the development of induction.

It is clear from the following that S4 has at least a general notion of *proposition-valued function* of integers.

> S4: You originally get a statement which says the $P(N)$. The actual statement, prove $P(N)$ true for N greater than or equal to some starting point.

More explicitly, it also appears in terms of a specific example of a function of this kind.

> S5: Prove by induction that $N!$ is greater than 2^N. So that's the first thing. You want to prove that $N!$ is greater than 2^N. Your $P(N)$ statement.

One construct that seems rather difficult for students is to see an implication as a total entity, a cognitive object. This rarely appears explicitly and there are some responses such as that of S6 which suggest that it is not present and as a result the student tries to prove $P(N + 1)$ rather than $P(N) => P(N + 1)$.

S6: I just try to remember that you prove it for $N = 1$ and then try to make a proof for $N + 1$.

This phenomenon seems to be widespread and tends to reappear when the student is confused by a particularly difficult induction proof.

Some students, such as S4, do appear to have made the encapsulation and speak of proving an implication.

S4: . . . you just pick a k and you have to show that, if its true for k, it is also true for $k + 1$.

Here is an example of a student who appears to have the schema of *implication-valued function.*

S7: . . . Using that, you provide a way of getting, proving every other answer after that. Like if it works for x then you prove it works for $x1$, no matter what the value of x is.
P: $x1$?
S7: $x + 1$.

Several students seemed to use an explicit *modus ponens* schema, for example,

S2: . . . starting out from $n = 1$, $n = 1$ implies 2, 2 implies 3, 3 implies 4, on up to 7.

We have already discussed the *explain induction* schema which most students had. On the other hand, there was no evidence indicating either that this schema was or was not encapsulated to a cognitive object that could be an aliment for *method of proof.* It is perhaps a subtle thing and it will probably be necessary to design questions aimed at eliciting this schema to be used in other studies. In any case, the students seemed to be using a definite procedure in dealing with the given problem. Four of them made this explicit and three others appeared to be using it. All but one dealt with the base case although three of them needed a little prompting. Five of the students indicated clearly that they needed to prove

$$(k + 1)! > 2k+1, \text{ given that } k! > 2^k$$

CONCLUSION

It seems clear from these results that, as a group, the students made considerable progress towards understanding mathematical induction. They were able to describe the process and discuss it meaningfully. Almost all of them were able to deal with problems in which they were asked to construct an induction proof.

They set up the arguments properly and most of them produced correct proofs. These problems were completely different from those that they had practiced with in class.

All of the schemas in the genetic decomposition appeared in the interview protocols, which suggests that this is a reasonable way to think about the student's construction of induction. Since most of the students solved the problem given in the interview, little can be said about how this was affected by whether or not the schemas appeared explicitly. Nevertheless, the one student, S3, who least appeared to possess these schemas was also the one student who did not solve the interview problem. For this reason, and because the overall class results were so satisfactory, it seems that attempting to stimulate directly the reflective abstractions that, according to our theory, are necessary to construct the concept of induction, is a useful method for teaching induction.

On the other hand, as we have seen above, there are many inadequacies in the present version and much needs to be done in order to improve the method and to evaluate its effectiveness. Certainly, the small class size of eight students might have been an important factor and additional studies need to be made with larger groups.

The instructional treatment can probably be improved by developing more effective techniques for inducing the reflective abstractions. This seems particularly important for implication-valued function and encapsulation of the explanation of induction. It would be nice to have activities that would more directly help the students put all the schemas together into a coherent explanation. At the moment, this is done with a traditional lecture which may not be enough.

Most classroom experiences do not bring students to the point of being able to even set up induction proofs properly. This may be why instruction on this topic is generally reduced to an explanation in lecture and lots of practice with variations on a small number of problem types, such as proving the formula for a finite sum. As methods such as the one described here are developed and become effective, it will be possible and necessary to introduce a greater variety of problems and to expect our students to learn to solve them. These should include much harder problems. One obvious class of problems involves situations in which proving $P(N + 1)$ is hard, even when $P(N)$ and $P(1)$ are known to hold. There are problems in which it is not easy to determine the first value of N for which $P(N)$ holds. There are even cases in which it is not clear (to the student) what N models in the problem (for instance proving by induction that a given relation is a "loop invariant" in a particular program) or what is the exact form of the statement to be proved.

In addition to these "internal" difficulties, we must learn how to help students understand an induction proof that appears in the context of some larger discussion. The ultimate goal, of course, is that proof by induction becomes an integrated part of a student's mathematics repertoire and that the student is more

or less able to decide when it might be useful to apply it in a given situation, without having been told beforehand that "this is an induction problem."

Of course, these are lofty goals, far beyond what is presently achieved, even with bright students (but without special talent in mathematics). Research such as described here, developing methods to attack the students' problem of constructing induction by attempting to influence directly what appear to be the actual mental processes used in this construction, seems to be useful and gives us hope that progress toward these lofty goals is possible.

REFERENCES

Dubinsky, E., & Lewin, P. (1986). Reflective abstraction in mathematics education: The genetic decomposition of induction and compactness. *Journal of Mathematical Behavior, 5*, 55–92.

Schoenfeld, A.H. (1983, April) *Theoretical and pragmatic issues in the design of mathematical "problem solving" instruction.* Paper presented at the annual meeting of the American Education Research Association, Montreal, Quebec, Canada.

Schwartz, J.T., Dewar, R.B.K., Dubinsky, E., & Schonberg, E. (1986). *Higher Level Programming*. New York: Springer.