TEACHING MATHEMATICAL INDUCTION II

Ed Dubinsky

This paper is a continuation of Dubinsky & Lewin, 1986 and Dubinsky, 1987. In the former, we attempted to formulate the beginning of a theory of learning abstract mathematical concepts at the post–secondary level. In the latter, we considered the question of applying this theory to Mathematical induction and discussed a prototype classroom implementation of a first version. Although the results were encouraging, we observed at the time that our instructional treatment of induction was incomplete in several ways.

In the present article we present, in detail, our full treatment of this topic, including the use of computer experiences with the programming language, SETL and its interactive version, ISETL. We describe two classroom experiments, Class I and Class II, in which the method was used and give the results of student performance in making proofs by induction.

Mathematical induction is a very difficult concept for undergraduates to learn. For many, the general activity of making proofs is unacceptable behavior — by their teachers as well as by themselves. Induction itself presents specific cognitive obstacles (which we have studied in the papers mentioned above) and students will continue to be unsuccessful with it as long as teaching methodology continues to ignore these difficulties. Finally, it is not the case that, once a student has grasped the idea of proof by induction, he or she will immediately succeed in using this method to prove all sorts of theorems.

What does seem to happen with our approach is that students develop a more positive attitude toward making induction proofs and, hopefully, proofs in general (we have even heard the word "fun" used in this context). They are totally successful in solving the straightforward, repetitive problems that are standard fare when induction is covered and are not always easy for students. When they are presented with problems that are different from what they have practiced with and/or present

difficulties peculiar to the specific problem, their success rate varies, apparently depending on their general intelligence, previous knowledge and how much time they have to spend on this particular activity. In general, however, they set up most problems correctly and it is usually clear that the student knows how to use induction and intends to do so. Specific difficulties of the particular problem may or may not prevent her or him from succeeding. This result is neither unexpected nor discouraging. We do not claim that our method will convert every student into someone who succeeds with every problem that uses induction. What we do claim to have developed and will describe in this paper is a theoretically–based instructional approach, using computer experiences, that transforms the issue of Mathematical induction from cognitive development to general problem solving. In other words, we feel that most students who have used our approach will be ready for an educational experience in problem solving that includes induction as one of the tools.

Something should be said about the "cost" of the method. The software is not a major expense. It runs on many mainframes and all but the smallest microcomputers. The real question is time. It is true that if the method is used only for Mathematical induction and the students must learn to program in ISETL then, although this is relatively easy, the total time spent on this topic will be considerably more than is usually given to induction. On the other hand, this method is applicable to other topics such as quantification (see Dubinsky, Elterman & Gong) and the study of the language can be integrated into an entire course in Discrete Mathematics as described in Baxter, Dubinsky & Levin. In this case, the course can cover most of the standard material in about the usual time and the portion of the course which is explicitly devoted to Mathematical induction is less than two weeks.

1 Theoretical Background

The theory on which our approach is based is the *genetic epistemology* of Jean Piaget. Our particular interpretation of this body of ideas is laid out in several papers such as Dubinsky & Lewin, 1986 and Dubinsky, 1987. Our discussion of it here will be no more than a sketch.

Mathematical knowledge consists of an interconnected collection of schemas corresponding to individual mathematical concepts. A schema is a more or less coherent set of cognitive objects and transformations by and of these objects. Roughly speaking, coming to understand a concept amounts to constructing one or more schemas corresponding to it. The construction process is called *reflective abstraction* and it includes a number of mental activities such as *interiorization* (the formation of an internal process corresponding to some mathematical transformation), *generalization* (assimilation of a new phenomenon to an existing schema), *coordination* (linking together two or more schemas), and *encapsulation* (making a mathematical object out of a cognitive process).

A critical question for education is the determination of what it is that makes this construction happen and how can we, as teachers, design activities that will help. Note that, already in this formulation, there is a departure from standard ideas about teaching Mathematics at the College level. Conceptual knowledge is not something that is transmitted verbally, or by example, from one who knows it to one who may or may not feel interested in learning it. The student must *construct* and this means that he or she must be cognitively active. Thus the challenge for the teacher is to design activities for students that will lead to them performing the reflective abstractions necessary to construct the concepts in question.

Piaget's general answer to this question lies in his concept of *equilibration*. The student responds to Mathematical situations by trying to assimilate them to her or his existing schemas. Sometimes this works, as when the student is asked to differentiate $x^{2.3}$. The student may have a schema for differentiating monomials with real exponents and may use it here in which case he or she is

successful. Or the student may be asked to determine the stability of solutions to a system of first-order differential equations with constant coefficients. If it is present and if it is used, the schema for finding the eigenvalues of a matrix and checking the real and imaginary parts will lead to the right answer.

Often, however, the student will use the wrong schemas, either by choosing unwisely or because the right ones don't exist for the student. Thus if the student is asked to differentiate 2.3^x one often gets 2.3^{x-1} for the answer. Or if the system of differential equations has variable coefficients, the method for finding eigenvalues does not work and the student may have no idea as to how to deal with the problem.

The key for Piaget's theory is that at this point it is necessary that the student be aware of a conflict or contradiction between the problem and the offered solution (or lack thereof). In attempting to transcend the contradiction, the student performs reflective abstractions to construct new schemas that will permit resolution of the conflicts.

There is much in this theory that can be very useful for the teacher to keep in mind. However there are a number of obstacles that must be overcome before it can be developed into an effective methodology. The first practical obstacle must surely be very familiar to anyone who has taught Mathematics. Students are generally quite satisfied with their answers and will usually struggle vehemently to convince the teacher (or whoever is determining the grade) that they are correct. Using grades to create the student's awareness of contradictions distorts the situation and may actually make it harder to get the student to develop an autonomous determination of the correctness of answers. Other, more effective, methods must be found.

A more serious obstacle, in our opinion, is that even when the student has reached the point of contradiction and there is an awareness that something must be done, some growth needs to take place; reflective abstraction does not always occur spontaneously. It seems that as the student

progresses through school and the Mathematics becomes more sophisticated, it becomes less and less likely that the reflective abstractions necessary to construct schemas for Mathematical concepts will be performed by the student without some sort of help. It is exactly this help that we believe is the main job of Mathematics education and our approach to teaching, or using a phrase that we prefer, *inducing learning* in Mathematics consists mainly of determining the reflective abstractions necessary to construct a concept and designing activities that will induce students to make them.

Easier said than done. A considerable amount of work with clinical interviews and theoretical analyses based on an understanding *on the part of the researcher* of the Mathematical content must be involved in determining what we have called the *genetic decomposition* of each topic. For getting the student to move along the path specified by the genetic decomposition, we have found computer experiences with the programming languages SETL and ISETL to be extremely effective. But the task is enormous and must be performed for each topic. Once completed, however, it seems reasonable to expect that it can be adopted by many teachers with a minimum of design effort mainly concerned with responding to local variations.

In the case of Mathematical induction the genetic decomposition which was worked out in Dubinsky & Lewin, 1986 and is shown in Figure 1 seems to be a reasonable description of one important path that students can and do take in constructing this concept. Work such as described here and in Dubinsky, 1987 can do no more than indirectly confirm this, since, after using our method, almost all of the students seem to have traversed the path, at least to the point of having acquired an explanation of the method of proof by induction which they appear to use on request when working problems. We present below some evidence in support of this contention.

INSERT FIGURE 1 ABOUT HERE

Designing the specific instructional treatments that will induce students to make the requisite reflective abstractions and move along the path specified by the genetic decomposition is more diffi-

cult. In Dubinsky 1987 we described a preliminary version of our treatment using the programming language SETL. In this paper we describe two subsequent experiments, one using SETL and one using ISETL. The students perform tasks on the computer consisting of writing and running programs in these languages. Our hypothesis is that this activity tends to lead them to construct the schemas which they need to put together in order to acquire the concept of Mathematical induction. Based on the results which are described below we feel that this phase is essentially complete and we have a viable treatment for Mathematical induction. It will be presented in textbook form in Baxter, Dubinsky and Levin.

2 SETL and ISETL

The programming language SETL is fully described in Schwartz, Dewar, Dubinsky & Schonberg, 1986. Its interactive version, ISETL is described in Baxter, Dubinsky & Levin. The main differences between the two versions are as follows. ISETL gives immediate values to expressions without the user having to wait for compilation; ISETL procedures, called `funcs`, are data and can be stored, passed as parameters, etc.; SETL runs only on mainframes but ISETL will also run on a number of microcomputers; SETL has much more extensive output facilities than ISETL; a number of features of SETL, such as backtracking and the data representational sublanguage are not implemented in ISETL. Although one of the two experiments described in this paper used SETL and the other used ISETL, it does not seem that the difference in language had much to do with the results. ISETL seems to be much more convenient to use in this context and that will be our choice in future work. Details of the language given in this paper will conform to ISETL usage in the few cases where they differ.

ISETL supports the basic simple data type and control features of an ALGOL–like language such as Pascal. Data types include integer, floating–point, Boolean, character string, tuple, and

(finite) set. Type and size of variables are dynamically determined so no declarations are required. In particular, a tuple may be considered as an infinite sequence, only finitely many of whose terms have been defined. Sets and tuples may be freely mixed and unlimited nesting is permitted.

A set whose elements are all tuples of length 2 is called a map and corresponds exactly to a relation (on finite sets) in Mathematics. If `F` is a map then `F{x}` is the image set. If this map is a function (single–valued to the right) then `F(x)` is its value at `x`. In this case, if the set `F` contains no pair (tuple of length 2) with `x` as its first component, then `F(x)` has the value `om` which stands for *undefined*.

The general syntax for forming sets is important. Initially a set can be formed by listing its elements and arithmetic progressions are possible, as in

```
{0,7..100}
```

which stands for the multiples of 7 from 0 to 100. A more powerful tool is the *set former* which has the general form,

```
{expr :  iterator list | condition}
```

Here `expr` is an expression, `iterator list` is a list of domain specifications for variables appearing in `expr` and `condition` is a Boolean expression. The entire expression forms a set by iterating through all values specified by `iterator list`, evaluating `expr` and placing the result in the set if `condition` holds, discarding it otherwise.

A `func` is an expression which can be assigned to a variable or evaluated as part of some other expression (including another `func`). Its syntax has the following form.

```
func(parameter);
    body of func including one or more return statements
end;
```

If a `func` is assigned to an identifier, say `F`, then the expression `F(expr)` is evaluated by first evaluating `expr` and then using the result as the parameter value in the boddy of the `func` to evaluate `F`.

3 Method of Instruction

Our instructional approach to Mathematical induction consisted of three parts. First, there was a number of computer experiences integrated with other topics in the course and taking place over a period of several weeks before the topic of induction was formally discussed in class. Second, about one–and–a–half weeks of class time was devoted to a discussion of induction. The final period was for students to work individually with induction and it formed the main basis for evaluation of our approach. We will discuss the three parts separately.

3.1 Preliminary activities

This was probably the most important part. It consisted of computer experiences designed to stimulate the students to perform the reflective abstractions to construct the required schemas that make up the genetic decomposition of Mathematical induction as shown in Figure 1. In most cases the computer experiences were integrated with other topics that were being studied such as sets, functions, propositional calculus and elementary proof techniques. Thus, in general, the tasks that were set for the students served more than one educational purpose.

Referring to Figure 1, our approach assumes that students have a concept of *function*. For induction, this concept will have to include at least the idea of quantities from one set being transformed into other quantities in a perhaps different set. For many students, it was in this course that they progressed beyond the very primitive idea of a function as an algebraic expression to be able to work with the dynamic aspects of this concept.

A second assumption is that there is a schema of *logical necessity* whereby it is understood that

if a certain statement is true then, a certain other statement is *of necessity* true as well. Again, to go beyond this to be able to perform logical operations with implications was something that many of the students had to learn in this course.

The first topic that we worked with was *method of proof*. After a student has written a simple program, for instance to test if two integers are relatively prime, then he or she may have interiorized the process involved in this determination and can think meaningfully about it to consider propositions such as; $n, n+1$ are relatively prime, $x, 2x$ are not, relative primeness is symmetric, it is not transitive, etc. This introduced direct proofs and (dis)proofs by counterexample. Later students worked with truth tables for implications to learn about proof by exhaustion, proof by cases and indirect proofs. They wrote programs that would generate the truth tables for complex propositions. Throughout it was pointed out that they were developing a repertoire of methods of proof. All of this was aimed at getting them to feel that nothing strange was happening on the day when they were told in class, "Today we are going to learn a new method of proof — Mathematical induction".

While all of this was still going on, the class was working with *proposition–valued functions*, sometimes, but not always with examples in which the domain was the positive integers. Pedagogically, it was very simple. It was enough to give them propositions (expressed in English) depending on variables and ask them to write and run ISETL `func`s that would evaluate the proposition when given values for the variables. This was extremely helpful in their study of propositional and predicate calculus. There was no difficulty in including examples of propositions which, later, would be given as theorems to be proved by induction.

Another useful task was to have them use their `func` to form a tuple of Boolean values (*true, false*) that represented the values of a proposition–valued function of the positive integers — at least for finitely many values. Here an important foundation was laid using the idea that an ISETL `tuple`

may be considered as an infinite sequence, but with only finitely many of its components assigned values. The ISETL quantity `om` which means *undefined* will be the value of any component which has not been defined. In the case of Boolean tuples, that is, tuples whose every component is to have a Boolean value, instead of *defined*, one can consider that the values of certain components are *known* while others have not yet been determined.

The next set of activities, overlapping in time the work with proposition–valued functions, concerned *implications*. In order to understand that the value of a function at a certain point in its domain could be an implication, it is necessary for the student to be able to think of an implication as an object. On the other hand, in order to be able to pass from $P(n)$ to $P(n+1)$ in an induction proof, the student must be able to treat an implication as a process, which we call *modus ponens*. Not only must both interpretations be possible for the student, but later, he or she will need to be able to pass easily from one to the other and, on occasion, interpret the same mathematical entity in two different ways at the same time.

Students who have been writing programs of any kind are probably familiar with `if - then` clauses which, as lines in a program, are objects and, in terms of flow of control, represent processes. It is useful to make this point for the student. We also used a large program to simulate a Medical Diagnosis system. A number of strings of the form `"A impl B"` were stored in a set. The values of `"A"` represented various diseases and the values of `"B"` referred to individual symptoms. The interpretation of the string was that if a patient had disease `"A"` then he or she displayed symptom `"B"`. The students were asked to write programs that answered various questions about the situation. Thus, when the student was thinking about storing the strings in sets, they appeared as objects. In other parts of the program it would be necessary to dissect the string and, based on information that `"A"` is true, pass to the conclusion that `"B"` holds. In this way, the object becomes a process. Taking the strings and combining them by transitivity to produce new strings requires the students

to revert, in their thinking about the implications, from processes back to objects.

We assume now that the student's concept of function includes the notion of a transformation that takes objects from one set and converts them into objects in another set. We also assume that both positive integers and implications are objects for the students and thus he or she is cabable of thinking about *implication–valued functions of the positive integers.* Of course, all of this is a repetition of the development of proposition–valued functions of which it is a special case. But it must be noted that the students are only likely to realize that implication–valued functions are a special case of proposition–valued functions *after* they have constructed both concepts. Thus we repeat some of the work of writing ISETL `funcs` that represent implication–valued functions of the integers and constructing `tuples` that represent the knowledge of the values of the function for a finite set of integers.

A key step occurs at this point. The students are asked to write an ISETL `func` that will accept a proposition–valued function P of the integers and return the *corresponding* implication–valued function Q. That is,

$$Q(n) = (P(n) \Rightarrow P(n+1))$$

The `func` is very simple. As it turns out, the input function can be in the form of either a `func` or a `tuple`. The solution that we expect is simply,

```
Implic := func(P);
              return func(n);
                          return (P(n) impl P(n+1));
                      end;
          end;
```

Writing this `func` and applying it to several of the proposition–valued functions functions that the students have worked with — in particular, several which are statements that will soon be theorems to be proved by induction — contributes to two very important accomplishments for the students. First, thinking about possible values for the components of `Implic(P)` gets them to begin

to interiorize the idea of determining that it has the constant value *true* — at least from some point on. This is exactly the idea of "proving the implication" which is so hard for students to think about in making induction proofs.

The second accomplishment does not really relate to induction directly and so will only be mentioned in passing. The idea that a `func` returns another `func` is profoundly disturbing for students. A procedure is supposed to return a value and, for most students, functions are not yet objects that can be discussed and returned as values. Indeed this particular example has two such instances, because not only the output of the `func`, but its input as well is a function. We will pursue this issue in another study which will be about functions.

Returning to induction, we see that at this point the student is ready to put it all together and construct the method of proof by induction. Two things are done to help the students with this task. First we assign a program which we call the *induction simulator*. Following is the description of what the students are asked to do. It includes a specification of the program and an explanation of how it is to be used.

1. Determine the precise statement of the proposition–valued function for the given problem and write a `func` called `prop_fn` that will evaluate it for any element of its domain.

2. Your program should begin by initializing an empty tuple, `P`. This means that we do not know if `P(n)` holds for any `n`.

3. Set up a loop in your program that runs `prop_fn` for `n` beginning at `1` and halts at the first `n` for which it returns `true`. This will be `n0` for this problem.

4. Write a `func, Modus` that will implement the implication-valued function corresponding to the proposition-valued function that you obtained in 1. That is, `Modus(n)` will be the value, in this problem, of `P(n) impl P(n+1)`.

5. Put the following loop in your program.

```
while ((P(n) /= om) and P(n)) do;
   P(n+1) := Modus(n)
   n := n+1;
end;
print("P is not proved for n = ", n);
```

Notice how closely the design of the program follows the steps of an induction proof. Of course, there will be some problems for which it is not so easy to see how to construct `prop_fn`, but the simulator is not to be considered as an important tool in making proofs! It is treated somewhat lighthandedly and the students are amused by the situation in which, if their program is correct, it will run forever. The main goal of this exercise is for the students to perform the reflective abstraction (described in Figure 1) of coordinating implication–valued function with modus ponens to obtain an explanation of induction. Then they are ready for a lecture on induction which essentially describes, exemplifies and ties together mental processes which, if our approach has been successful, the students have already constructed.

3.2 Explicit treatment of induction

The classroom treatment of induction takes about $1\frac{1}{2}$ weeks. It consists of a relatively traditional treatment of the topic in which a general description of the method is presented in lecture form, interspersed with the working of an example, followed by the students working problems in class and receiving immediate feedback in the form of a subsequent explanation of how to do each problem and a discussion of errors that may have been made.

There are only two points of novelty that are worth mentioning. First, the explanation of induction follows very closely the cognitive development that we have described in the preceeding pages. It consists of the following four steps.

1. Express the problem as a proposition–valued function of the positive integers P with the idea that the goal is to show that this function is eventually constant with constant value *true*.

Determine what, in the problem is represented by the independent variable n and determine the exact meaning of the proposition $P(n)$.

2. Determine the base case n_0 and establish $P(n_0)$.

3. Derive the corresponding implication-valued function Q and determine the exact meaning of the proposition $Q(n)$.

4. Prove that from the base case on, Q has the constant value *true*. Here there is some discussion about reducing the expression for $P(n+1)$ to the expression for $P(n)$ and/or the use of $P(n_0)$ and $P(n)$ to establish $P(n+1)$.

The hope is that most of the discussion about the above four points will appear to the students as pointing out and *institutionalizing* things that they already know. In the best of situations they will be excited by the realization that they can now solve problems which only a few weeks ago seemed to be way over their heads. In the worst case, they understand that Mathematical induction is a very difficult method and they have acquired some idea of how to do it.

The other point which is mildly novel is that instead of the traditional use of metaphors like *ladders* or *dominos* we refer to moving along a `tuple`. That is, one wishes to move along the `tuple` which represents P, inserting the value *true* at each step. There is a *true* at a certain component of the tuple for P and also at the corresponding component of the `tuple` representing Q. These two *true*s permit one to move one step further and insert *true* at the next component of the tuple for P.

Is this a metaphor? Or are these tuples real for the student and thus the method of induction becomes for them a recipe for an activity that is actually performed in each proof? This is a very important psychological problem which should be studied. Our guess is that the tuples are somewhere in between metaphor and reality and it is this feature that represents at once the uniqueness and the power of our way of using computer experiences.

3.3 Drill and Practice

After the classroom discussion of induction, students in both classes that we are discussing were given the same set of 10 problems to solve and submit. They had about a week to do it and were asked to work entirely on their own without interaction with other students and without use of any source material other than their classroom notes and the textbook for the course. The purpose of this assignment was to give them an opportunity to work fairly intensely on a wide variety of induction problems. The 10 problems are given in Table 1.

(INSERT TABLE 1 ABOUT HERE)

It is interesting to compare these problems with the examples which the students studied in class and which, for the most part were solved, either by them or by the instructor before they were given this assignment. These are given in Table 2. As far as could be determined, other than some formulas for finite sums, these problems include every induction problem in the experience of these students. If there were others, the students showed no hint of memory of them.

(INSERT Two-column page ABOUT HERE WITH FOLLOWING FOOTER)

Table 2. Induction examples worked in class

The two classes did not study exactly the same problems. Class II came one semester after Class I and some attempt was made to improve the choice of examples. In particular, divisibility problems were omitted from the examples studied by Class II, an example involving inductive definition was included, and the problem on convex polygons was omitted. Also, Class II worked one problem that eventually appeared as number 7 on the assignment. The purpose of this was to isolate the case of *minimal learning*.

In Class I the treatment of induction occured about half way through the term and so there was an opportunity to apply induction in other contexts. Theorems about graphs and trees as well as inductive definitions in Finite State Automata were topics in which induction was used. No

evaluation data was gathered, but the students seemed comfortable with it and on occasion were able to think of using induction when it was not the obvious thing to do. Because of scheduling restrictions the treatment of this topic in Class II came at the end of the course so there was no opportunity for applications.

4 Students

Class I took place during the spring semester, 1986 at University of California, Berkeley in an experimental section of a course in Discrete Mathematics. There were 24 students, mainly sophomore Computer Science or Computer Engineering majors. Their average Math SAT score was 720.

Class II took place during the Fall semester, 1986 at Clarkson University in a single section course in Discrete Mathematics. There were 16 students, mainly sophomore Computer Science majors. Their average Math SAT score was 657.

An attempt was made to find out something about the students' previous contact with Mathematical induction. In Class I a questionairre was distributed. Five students said that they had no previous experience, 10 said that it was very brief and 9 said that they had some, but not a lot. About half of the students said that they had succeeded at least once in the past to make an induction proof on their own, but not all of them could recall anything about the nature of the problem. Altogether 6 students were able to remember something about proving summation formulas; two said that they had used induction in studying binary trees and one remembered something about proving recursive programs correct. A classroom discussion indicated that induction was not a technique with which they were very familiar. In a second questionairre given after the course had covered induction, the students expressed the opinion that they did not know very much about induction before this course.

In Class II, a classroom discussion indicated even less familiarity with induction. These students

were less advanced than the 18 students studied in Dubinsky & Lewin, 1986. It will be recalled that that in this group of Juniors, Seniors and some graduate students, the students' understanding of induction was very weak both before and after the (traditional) instruction.

5 Results

The two instruments of evaluation were the students performance on the assignment given in Table 1. and their responses to the questions listed in Table 3. These questions were given to the students to complete during class. They are exactly the same questions as were used in the interviews in our previous two studies of Mathematical Induction reported in Dubinsky & Lewin 1986 and Dubinsky 1987.

(INSERT Table 3 ABOUT HERE)

The assignment shown in Table 1 was graded by a graduate assistant and subsequently checked by the instructor. Where the instructor disagreed with the assistant the score was changed. This resulted in a total change of less than 1%. The average scores are given in Table 4.

(INSERT Table 4 ABOUT HERE)

Of the 400 problems assigned to the 40 students, 213 of them were done correctly. The errors in the remaining 187 problems group into the following five categories.

a. Unable to determine the proposition-valued function P and/or incorrect or missing interpretation of the value of $P(n)$.

b. Omits consideration of the base case.

c. Sets up induction proof properly but does not succeed in proving that $P(n) \Rightarrow P(n+1)$.

d. Miscellaneous error that is serious but does not make the entire solution worthless.

e. Problem omitted or nothing much of value in the solution.

The frequency of occurrence of these various errors for each problem and for the total assignment is shown in Table 5. The two classes are considered together. There were 9 cases in which one problem had two errors so the total number of errors is 196 rather than 187.

(INSERT Table 5 ABOUT HERE)

The results on the questions in Table 3 are best described individually. All but 2 of the 40 students turned in these essays. On Question 1a, 28 of the 38 students indicated that they had an idea of method of proof as a general tool. They gave at least one example other than Mathematical induction. Another 6 students were able to write about a method only in terms of induction or some portion of it such as "showing that a proposition–valued–function is always correct". The remaining 4 students did not seem to grasp this concept.

On Question 1b, 28 students gave a description in terms of a transformation from positive integers to propositions or the two–element set {*true, false*}. In addition, 6 other students gave a satisfactory answer but felt compelled to include an explicit formula for a function. In their cases it seemed that they may not have totally freed themselves from thinking of functions as formulas. The remaining 4 students seemed confused on this point.

There were 29 correct answers on Question 1c, 33 correct answers on Question 2, and 27 correct answers on Question 3. For the most part the student could give a correct response by repeating from memory things that had been said in class.

The last one, Question 4, was the most interesting. The responses of 22 students included an explicit iteration of Modus Ponens, coordinated with evaluation of the implication–valued function. They began at 1, continued for a few steps that they described completely and jumped to 10. Several did the same for 1000, others simply pointed out that it was the same thing. This way of thinking about induction had only been mentioned very briefly in Class II and not at all in Class I. Another 11 students simply repeated or rephrased the principle of Mathematical induction. The remaining

5 students gave incomplete, confused, or wrong answers.

6 Discussion

These results confirm the initial indication in the prototype study reported in Dubinsky, 1987 and are in stark contrast to what one usually expects when Mathematical induction is taught in the traditional manner. Only a small percentage of the 18 students (mainly Juniors and Seniors) whose performance was discussed in Dubinsky & Lewin, 1986 were able to explain induction reasonably or use it to set up the solution to a problem, much less complete it.

In the present case, Table 4 suggests that almost all 40 of these students are able to use Mathematical induction to solve problems. As a group their performance was excellent on Problems 1,2,3,4 and 7. On the other 5 problems the results are not so strong but still more than acceptable. If both classes are considered together, there is really no problem that was too difficult for them. The weakest performance was on Problem 6, but a worst score of 5.6 out of 10 is not so bad. If one looks at the two classes separately, then one might be a little concerned about the performance of Class II on Problems 5 and 8, but still the overall average is impressive.

The results appear even stronger if one begins to analyze individual questions. One cannot say that Problems 5,6,8 and 10 are easy. Indeed, on Problem 6 the students' inexperience with unpacking the digits of an integer (an echo of the younger child's difficulty with the concept of place value?) could have been a factor and in Problem 8 the difficulty of figuring out a general rule stopped many students before they had a chance to use induction.

The analysis of errors in Table 5 tells us more. Of the total of 196 errors, 78 occurred after the induction had been set up properly and had only to do with the specific difficulty introduced by the particular problem. Another 18 were omission of consideration of the base case. Since this number is so small it does not suggest a pattern and might be explained as oversights. Thus it could be argued

that only 100 errors had to do with induction per se and that in 75% of the cases the students were able to apply induction correctly to attack the problem and in 53% of the cases (213 out of 400) they were successful in completing the solution.

A final question on the nature of the problems and the students' solutions concerns the extent to which their performance represented *transfer*. That is, a major goal for Mathematical induction (as with any general method) is that the students develop a general "high–level" understanding of this method as a concept and are not only able to solve problems very similar to the ones they practiced on or whose solutions were shown to them, but can also apply the method to problems and situations which are somewhat different and even completely novel. A second goal is that students ultimately become capable of choosing Mathematical induction out of a repertoire of approaches that they possess in situations where, other than the nature of the problem itself, there is nothing to suggest induction as opposed to any other method.

Our approach has little to say at this point about the second goal but our results do relate to the first. Comparing Tables 1 and 2, it is clear that Problems 3,4 and 7 of the assignment are quite similar to examples with which the students in both classes were familiar (see examples 2,3 and 6 in both columns of Table 2). Indeed, Problem 7 on the assignment is identical to Example 6 for Class II in Table 2.

If we consider Problems 1,2,6,8 and 10 of the assignment, however, there are some connections, but these problems are not exactly the same as ones the students have seen. For instance, Problem 2 of the assignment is similar to Example 2 for both classes, but the student still must obtain the closed form or think about the string of additions. Problems 1, 6 and 10 were familiar for the students of Class I only in that they had to do with divisibility which appeared in Example 4 for Class I. One could perhaps argue that parity involves divisibility, but this hardly suggests that familiarity with Example 4 for Class II helped with Problems 6 and 10. Finally, Problem 8 becomes similar

to Example 2 only after the student has succeeded in expressing the general situation, which was a major stumbling block for many.

The remaining problems, 5 and 9 for Class I and 1,5,6,9 and 10 were quite different from anything the students had seen as problems to solve and their performance on these could be an indication of transfer of knowledge taking place.

One can compare these comments with Table 4 and see that there is a rough correspondence between performance and special difficulties such as transfer, place value, and finding general formulas. The mix, however, is too complicated (and unavoidable) to permit looking for a more precise relationship. These difficulties might however help explain the difference in performance between Class I and Class II which, though not excessively large, is noticeable and should be considered. There are two major differences between the two groups that are easy to discuss. One is the average SAT scores of the classes (720 vs. 657) and the other is the difference in familiarity of the problems for the two classes. Both differences favor Class I and this might explain their higher scores.

Turning to the results on the Questions in Table 3, we have little to add. These kinds of questions are useful for analyzing the difficulties of students who have not yet mastered induction. The fact that the overwhelming majority of students were able to answer these questions satisfactorily could be due to their understanding of Mathematical induction, but it could also be a case of remembering what they had heard in class. Or it could be a mixture of the two because memory in a context where there is understanding is different from rote memory of comments about an incomprehensible idea. All that we can say with certainty is that the results on these questions do not suggest that the students are having difficulty in thinking about Mathematical induction and the schemas of which it is composed.

7 Conclusions

If one accepts tests results as indicative of students' knowledge and if one admits a take–home assignment as a valid test, then the results reported in this paper indicate that the students in the two classes learned how to use Mathematical induction to prove theorems. As far as we can tell from the questions they were asked, the students appeared to have passed through the genetic decomposition of induction shown in Figure 1 at least to the point of being able to apply induction. Their lack of difficulty with either problems to solve or questions to answer concerning the schemas which appear earlier in the decomposition is consistent with our hypothesis that the computer experiences helped them to construct these schemas. Their overall performance on induction problems suggests that our approach was effective in bringing them to this point.

These results indicate that, for these students, Mathematical induction is a concept that they have acquired and they are now ready for various kinds of learning experiences that use induction to solve problems and construct other concepts.

The method that we have used here is now developed to the point were it need no longer be considered experimental and can be used in ordinary classrooms when a minimum of computer facilities is available. Preparation required by the teacher is not excessive and full details on what needs to be done are available from the author.

Several things remain to be seen. To what extent will students who have used this method to learn induction retain their knowledge over a period of time? How will they respond when called upon to apply induction in a variety of Mathematical contexts? Will they think of using induction on their own in situations when it is not explicitly suggested by the teacher or the problem? Will the method continue to be as effective as reported here when it is used by other people and after time and repetition have eliminated the "Hawthorne effect"?

We have no answers to these questions at this time. We can only say that the students who have

passed through this experience appear to be, as a group, better equipped to deal with Mathematical induction than any group of undergraduates we have ever come in contact with.

Acknowledgements

The author would like to thank the Mathematics Department of the University of California, Berkeley for the opportunity to conduct Class I and also Michael Clancy and Alan Schoenfeld for many useful discussions regarding this work. A number of people assisted in various ways. These include Paul Blizniak, Manuel Bronstein, Flor Elterman, Cathy Gong, Doug Smith and Norbert Vogl. Perhaps the most important contribution was made by the students in the two classes who responded to the ideas, the huge amount of work and the opportunity to learn with enthusiasm, skill and good cheer.

REFERENCES

Baxter, N., Dubinsky, E. and Levin, G., (in preparation) *Discrete Mathematics.* New York: Springer–Verlag.

Dubinsky, E., and Lewin, P., Reflective Abstraction and Mathematics Education, (1986) Journal of Mathematical Behavior, 5,1.

Dubinsky, E., (1986), On Teaching Mathematical Induction, I, Journal of Mathematical Behavior, (to appear).

Dubinsky, E., Elterman F., and Gong, C., (1986), The Student's Construction of Quantification I, (submitted for publication).

Schwartz, J., Dewar, R., Dubinsky, E. and Schonberg, E. (1986). *Programming with Sets: An Introduction to SETL.* New York: Springer–Verlag.

Use induction in each of the following problems

1. Show that
$$(n^4 - n^2) \bmod 3 = 0$$

2. Prove that
$$(1)(3) + (2)(4) + (3)(5) + \ldots + n(n+2) = \frac{n(n+1)(2n+7)}{6}$$

3. Show that
$$\sum_{i=1}^{n} \frac{1}{i(i+1)} = \frac{n}{n+1}$$

4. Consider the following SETL procedure.

```
proc compute(x,y);
        z := x;
        w := y;
        (while w > 0)
              z := z + y;
              w := w - 1;
        end while;
        return z;
end proc;
```

   Show that the following expression is a loop invariant for the beginning of the while loop.
$$yw + z = x + y^2$$

5. Show that in a casino with chips worth \$5 and \$9, any sufficiently large amount of dollars can be represented.

6. Show that an integer consisting of 3^n identical digits is divisible by 3^n.

7. Show that the following inequality holds for n sufficiently large.
$$n - 2 < \frac{n^2 - n}{12}$$

8. Formulate and prove a general formula stemming from the following observations.
$$\begin{aligned} 1^3 &= 1 \\ 2^3 &= 3 + 5 \\ 3^3 &= 7 + 9 + 11 \\ 4^3 &= 13 + 15 + 17 + 19 \end{aligned}$$

9. Let Σ be an alphabet and A a language in Σ^*, with the property that $A^2 = A$. Show that $A^* = A$.

10. Show that $11^{n+2} + 12^{2n+1}$ is divisible by 133.

Table 1. Induction Assignment

CLASS I

1. Show that for n sufficiently large,

$$n! > 4^n$$

2. Show that

$$\sum_{i=1}^{n}(3i^2 - 3i + 1) = n^3$$

3. Consider the following SETL procedure.

```
proc huh(x);
      $ x a positive integer
      z := 1;
      c := 1;
      (while c < x)
          z := z + 2*c + 1;
          c := c + 1;
      end while;
      return z;
end proc;
```

   Show that the following expression is a loop invariant for the beginning of the while loop.

$$z = c^2$$

4. Show that the sum of the cubes of 3 consecutive positive integers is divisible by 9.

5. Show that the sum of the interior angles of a convex polygon of n sides is $(n-2)\pi$.

6. Show that for n sufficiently large,

$$\sum_{i=1}^{n}\frac{1}{\sqrt{i}} > \frac{19}{10}\sqrt{n}$$

CLASS II

1. Show that for n sufficiently large,

$$2^n > n^2 + 2n - 2$$

2 Show that

$$\sum_{i=1}^{k} i^3 = \frac{k^2(k+1)^2}{4}$$

3. Consider the following SETL procedure.

```
proc huh(x);
      $ x a positive integer
      z := 1;
      c := 1;
      (while c < x)
          z := z + 2*c + 1;
          c := c + 1;
      end while;
      return z;
end proc;
```

   Show that the following expression is a loop invariant for the beginning of the while loop.

$$z = c^2$$

4. Let (F_n) be a sequence of integers defined by

$$F(1) = F(2) = 1,$$
$$F(n+2) = F(n) + F(n+1), \qquad n > 0$$

   Show that every third term is even and the other terms are odd.

5. Show that for any set of binary words of length n there is an algorithm for iterating through all of them without repetitions changing exactly one bit each step.

6. Show that for n sufficiently large,

$$n - 2 < \frac{n^2 - n}{12}$$

1. Describe each of the following terms and give an example for each.

   a. Method of proof

   b. Proposition–valued function of the positive integers

   c. Implication–valued function of the positive integers

2. Explain, in your own words, what it means to make a proof by induction. What is involved in making such a proof?

3. Modus Ponens (method of the bridge) refers to the mental activity by which, if you know an implication $A \Rightarrow B$ and you know that A holds, then you know that B holds. What role does this concept play in Mathematical Induction?

4. Suppose you have applied the method of Mathematical Induction to some statement, starting, say, with the first value equal to 1, and successfully completing the proof. Having done this, what would you say to someone who didn't know much about mathematics to convince her or him that, because of your proof, the statement is true for a fixed integers, say $n = 10$? What about $n = 1000$? Give the most concrete way that you can of trying to convince the person.

Table 3. Questions about Induction

	Questions										
Class	1	2	3	4	5	6	7	8	9	10	All
I	8.3	8.9	8.8	8.2	6.5	5.9	7.2	7.1	6.1	7.2	74.1
II	6.9	9.4	9.2	7.4	4.8	5.3	8.7	4.3	5.1	5.2	66.3
Both	7.8	9.1	8.9	7.9	5.8	5.6	7.8	6.0	5.7	6.4	71.0

Table 4. Results of Induction Assignment
Average score by class and by question (maximum 10 points per question)

	Questions										
Class	1	2	3	4	5	6	7	8	9	10	Total
a	0	0	0	1	3	0	0	1	0	0	5
b	1	2	5	6	2	0	1	0	0	1	18
c	11	2	1	0	10	23	2	15	1	13	78
d	2	0	0	5	6	1	5	0	17	0	36
e	2	0	0	2	8	5	5	9	18	10	59

Table 5. Frequency of Error Types