An Abstract Algebra Story

by

Uri Leron and Ed Dubinsky

INTRODUCTION

Statement: *The teaching of abstract algebra is a disaster, and this remains true almost independently of the quality of the lectures.*

> We agree.
> And we think there's a fairly wide consensus on this among experienced abstract algebra instructors, and an even wider one among experienced students.

Statement: *There's little the conscientious math professor can do about it. The stuff is simply too hard for most students. Students are not well-prepared and they are unwilling to make the effort to learn this very difficult material.*

> We disagree.
> But we suspect that many experienced abstract algebra instructors hold such beliefs. This is especially true for some excellent instructors: Their lectures are truly masterpieces, surely you can't improve much on *that*; so if the students still fail, that's too bad, but it can't really be helped[1].

We claim that, far from being an immutable fact of nature resulting from inadequacies of the student, this failure is, at least in part, an artifact of a too narrowly conceived view of instruction. In fact, *replacing the lecture method with constructive, interactive methods involving computer activities and cooperative learning, can change radically the amount of meaningful learning achieved by average students.*

In this paper we would like to paint a picture of such an alternative approach, which we and others have been developing and using in our classes over the last several years. We are painfully aware of the limitations inherent in any attempt to give such a description by means of the written text only. It would have been much better if you could actually visit our classes and observe the dynamics of the students' interactions with both the computer and their peers. By way of compromise, we will try to simulate such a visit by organizing our paper around four classroom "scenarios" and some commentary on the events depicted in each scenario. As

---

[1] Relating this to the Monthly Editorial of May 1992, we are not here blaming the math professor for this state of affairs. In fact, one of our major claims is that, without a major change in teaching *methods*, even the best lecturer with the best of intentions cannot do much to improve the learning of abstract algebra for the majority of students.

a matter of principle, we have tried to make the scenarios as realistic as space limitation permits.

First Scenario: What Is It All About?

Background.

Our approach involves students working on computers in laboratory sessions and on their own. It also involves classroom discussions and assigned exercises, mainly of the paper and pencil variety. The students work in teams which for the most part remain fixed for the entire semester.

This scenario takes place very early in the semester, after the students have done some computer work. For example, in the exchange we are about to describe, the students are using a number of programs they have written to implement the group axioms. These are listed in Appendix 1. The students wrote these programs after very brief and intentionally vague discussions (in the text and in class) of properties of binary operations.

Reality.

You enter an abstract algebra class. The class is taking place in a microcomputer lab. The students are working on computer activities in teams of 2 to 4. The expressions they type at the keyboard, except for some minor details, look pretty much like standard mathematics; as a mathematician you have no trouble understanding the meaning of what they type, though you may be totally unfamiliar with computers. Before hitting the Return key, which would instruct the computer to evaluate their expression and exhibit the result, the students engage in a lively discussion trying to predict what this result is going to be. For example, you might observe something like the exchange in Figure 1[2] .

(Editor: Dialogues such as the following should be placed in a box in a convenient place relative to the text.)

Figure 1: Student interactions within a mathematical computerized environment.

Note: the students' input is preceded by the '>' prompt; the computer's output is not

[2] Real exchanges are characterized by a lengthy and messy zigzag path, and their transcription would take up more space than we can afford in this article. Therefore we are limited to bringing here only a stratified and compressed version of a real exchange. In fact we can include only brief, isolated fragments.

Interaction with the computer	Interaction within the team
> Z12 := {0..11};	Why doesn't Z12 have 12 in it?
> a12 := |x,y -> (x+y) mod 12|;	What's the difference between Z12 and a12?
> is_group(Z12,a12);	What are the inputs to is_group?
true;	
> identity(Z12,a12);	What's happening here? I thought it would be true or false? What is *identity* supposed to return?
0;	
> is_group(Z12-{0},a12);	What's wrong? Let's check the group properties.
false;	
> is_closed(Z12-{0},a12);	
false;	Aha! So it's not closed, but I can't see why. Let's try some numbers.
> 7 .a12 8 in Z12-{0};	
true;	
> 7 .a12 5 in Z12-{0};	
false;	I know!
	Let's try another operation.
> a := |x,y -> x+y|;	
> is_group(Z12,a);	
false;	
> 7 .a 8 in Z12;	
false;	It's because 7+8 = 15, not 3.
	I can write it like this.
> m12 := |x,y -> (x*y) mod 12|;	What do you think about times?
> is_group(Z12,m12);	I think it will also be a group.
false;	Oops!
> is_closed(Z12,m12);	
true;	So it is closed...
> is_identity(Z12,m12,1);	
true;	... and it does have an identity...

```
>        has_inverses(Z12,m12);
false;                                  ... but no inverses! What's an inverse,
                                        anyway?
>        is_invertible(Z12,m12,0);
false;                                  Oh! It must be zero. Let's take it out.


>        is_group(Z12-{0},m12);
false;                                  Oops! I don't understand it
>        is_closed(Z12-{0},m12);
false;                                  Not closed? But is was closed
                                        before... Must be the zero...


                                        What about this...
>        2 .m12 6 = 0;
true;
>        2 .m12 6 in Z12;
false;                                  that's it. Let's try another mod? 11?


>        Z11 := {0..10};                Okay, this is the set...
>        m11 := |x,y := (x*y) mod 11|;  ... and the operation
                                        now let's try it. I think it will work.


>        is_group(Z11,m11);
false;                                  Oops! Oh, yes, the zero again.


>        is_group(Z11-{0},m11);         Now try it.
true;                                   Yeah!!!
```

Reflections.

At this point you, the reader, like the students in our story, have surely formed some hypotheses about what is going on here. Unlike our students, however, your hypotheses cannot be run on the computer for comparison and checking, so let us now try to attend to some possible interpretations (and mis-interpretations) of what is really going on here. We present some likely hypotheses and questions posed by an idealized reader (**IR**), each followed by our reaction. In fact, these include some of the questions we have most often been asked when discussing our method with colleagues.

Idealized Reader (conjecturing): *The students are involved in a mathematical investigation concerning group-theoretic properties of modular arithmetic systems, also practicing at the same time their knowledge of the group axioms*.

True.

IR: *The students are learning by the "discovery method"*.

This is only partially true. In the first place, the students are indeed involved in posing questions and in trying to discover answers, but far from being left on their own to do that, they are guided by a worksheet provided by the instructor. For example, the activity in the above exchange might evolve in response to the following task:

Explore the modular systems Z_n (with or without 0) relative to addition and multiplication mod n. Formulate some conjectures, test them and try to come up with some explanations.

In the second place...

IR: *...But how can you expect students to discover in a few hours what took the best mathematical minds centuries?*

I was just getting there... In the second place, the students are *not* expected to actually arrive at complete, "correct" answers. The main purpose of the activities is to give them an *experiential basis* to which they can later relate the more abstract and formal treatments. Thus, subsequent discussions of a mathematical concept are more meaningful for students who have made a non-trivial effort trying to figure it out on their own, whether they have actually discovered it or not.
Rather than "discovering" mathematical concepts, we think of our students as *constructing* them (in their mind). The computer experiences and classroom discussions are meant to help them make these mental constructions. This important distinction will come up again on several occasions in the remainder of the paper.

IR: *The computer system the students are working with is some elaborate CAS (Computer Algebra System), equipped with special software dealing with group theory.*

Wrong. They are programming in a computer language designed to be as close as possible to the standard language of written mathematics. In fact, had you come to visit a week earlier, you'd find the students involved in *programming on their own* the functions is_group, identity, is_closed, has_inverses, etc. (see Appendix 1.) The computer language they are using is called ISETL (for Interactive SET Language).

IR: *Programming on their own? Doesn't learning how to program take a lot of time away from learning the mathematics in the course*?

Because the syntax and basic constructs of **ISETL** are so close to those of standard mathematics, learning the language is inseparable from learning the mathematics -- the programming "overhead" is minimal. You can check this for yourself by seeing if you have any trouble understanding the ISETL expression in Figure 1. With our students, we spend only the first few sessions actually dealing with the language, and even then we do quite a bit of relevant math (like properties of the modular operations and of permutations). And, of course, even this small overhead is incurred only in the first course using our method. See the appendices for the actual code written by the students, and observe that in writing this code, and in using it to explore particular groups, the students gain an understanding of the group concept that is of a quite different sort from the one gained by listening to lectures and doing paper and pencil exercises (or even by using the computer for drill and practice).

IR: *Still, wouldn't it be better if you gave them a software package in which functions like is_group, identity, has_inverses are pre-programmed? This way you'd save them the time of learning to program and the time of programming all these functions on their own, and still they would be able to "interact" with the computer.*

Programing is_group and the other functions is where the most important learning occurs. It is *the goal*, not just a *tool*. Experience, as well as modern learning theory (**s**ee, for example, Davis, Maher and Noddings, 1990 or Selden and Selden, 1990) tells us that one doesn't learn the group concept by memorizing the definition. In order to acquire meaning, the group concept has to be *constructed* in the learner's mind. Our method is based on the premise that if the students are asked to construct the group concept on the computer (by programming it), there is a good chance that a parallel construction will occur in their mind. Or, to use Seymour Papert's metaphor, the students learn best by *teaching the computer*. (Papert, 1980. See also, Clement, Lochhead and Soloway, 1980)

IR: *So this method is not specific for the teaching of abstract algebra only*?

Right. Learning mathematics is always about constructing mathematical processes, objects and relations among them. The ingredients for these constructions are always sets, functions, logical expressions (such as quantifiers) and the results of previous constructions. **ISETL** has been especially designed to express naturally and clearly this kind of construction.

IR: *Is there any experience? Data*?

There is a rather substantial experience in teaching discrete mathematics, calculus and abstract algebra with our method. A number of research papers have been published, describing these experiments, and text books on all three topics have been, or are about to be published (Baxter, Dubinsky and Levin, 1988; Dubinsky and Schwingendorf, 1992; and Dubinsky and Leron, in preparation). Changing from a more traditional pedagogy to using such a radically different method requires a considerable initial dedication and enthusiasm by the instructor, but the results are very encouraging.

IR: *Are the computer activities meant to replace the traditional lecture?*

Yes. We think that the lecture method is, for most students, quite ineffective. Furthermore, it makes them feel stupid, alienated. Most students would tell you (we actually took the trouble to ask them!) that they have very little idea of what the lecture is about, even when it is delivered by a master lecturer**.** And if you try, say a year later, to see how much was retained from the course, you'll discover that it is close to nothing. (See, for example, Vinner, 1992.) In a sense this quantity might even be considered less than nothing, for many of these students have come to hate this beautiful piece of mathematics.

IR*: Are you assuming, then, that through the activities they get all the instruction they need?*

Not at all. The activities are followed by team work on assignments, team discussions, class discussions of subtle points, summary handouts (or assigned reading) of definitions and proofs, exercises etc. The role of the computer activities is, as we have said before, to provide an *experiential basis* for all the other learning modes. An important key to making this work is that students need to *reflect* on the computer (or paper and pencil) constructions that they make. A powerful stimulus for reflection is our insistence that they do their work in teams.

Discussing and explaining what they are doing can lead to mental constructions that are parallel to the ones they are making on the computer.

When the students eventually come to learn the "official", general, abstract, formal version, this is perceived by them not as totally strange and prohibitive (as we believe is the case in standard lectures, where such abstractions are presented without any experiential basis), but as an elaboration of their previous experience. In popular terms we may say that the activities provide an initial intuitive familiarity with the topic to be learned. In more psychological terms (supported by an elaborate theoretical framework and research), we may say that the activities help the students to "construct" the mental processes, objects and relations necessary for a meaningful understanding of the topic.

IR: *I, too, believe in giving students intuitive explanations (such as the intuitive idea behind some complicated proof), which I always add to the formal part of my lectures. So why all the fuss?*

Experience, theory and research all point to the fact that verbal explanations of issues that do not relate to the student's prior experience are quite ineffective (except for a few individuals with special talent in mathematics, which naturally includes the readers of this journal....) Intuition is the result of personal experience based on activity and interaction. A verbal representation of *your* intuitions (based on *your* past experience, activity and interaction) usually fails to re-create the same intuitions in the student's mind. Students need to construct the experience for themselves. Verbal explanation can help if they come *after* the student has formed an experiential basis. For then they serve to elaborate and conceptualize something the student has already vaguely known through the experience. We may say that verbal explanations can elaborate and explicate existing intuitions, but cannot *create* new ones.

IR: *Is your method so perfect? Surely there must be some difficulties, some unsolved problems, some things you are not so sure about...*

You must be kidding! The traditional method of teaching by lectures and exercises has developed over several hundred years. We feel that we have an approach which represents a significant improvement, but we are just beginning. Although we have been able to implement our method to obtain promising results in our own classes, and we are even beginning to learn how to disseminate this approach to others, a multitude of problems remain and it will be a very long time before they are all solved.

There are difficulties inherent in the use of a computer. It is slow (as of this writing, we are barely able to work on the latest PC with the group of permutations of five elements); it is not easy for a computer to deal with infinite objects; the computer can only help indirectly with making proofs; and it is not always convenient for students to work in a computer lab. There are difficulties connected with students working in teams. Personality conflicts arise; students tend to overuse the "divide and conquer" method as opposed to teamwork; and there are inconveniences connected with teams getting together outside of class. Finally, we cannot overemphasize the profound change in the teacher's attitude and the need to develop new skills that are part of adopting this method. Some sort of re-training is necessary.

Nevertheless, our theoretical perspective, experiences and research, and those of others, as well as reports from our students, all convince us that the improvement --- the revolution! --- in student learning justifies the effort.

Second Scenario:
Constructing Lagrange's Theorem and its Proof

Background.

This topic is treated about one-third of the way into the semester. At that time the students, as well as the software environment, have grown considerably smarter. In particular, we describe three points which are necessary for understanding the next scenario.

1. At this point, the students are well-acquainted with groups, subgroups and cosets, as well as with various examples, notably the modular groups Z_n (with addition mod n) and the symmetric groups S_n. All this knowledge is represented in ISETL by various functions and other mathematical objects which the students have constructed in previous activities. These are collected in a special initialization file (called *isetl.ini*) on the computer's disk (ISETL's long-term memory), and is automatically loaded each time ISETL is loaded. Thus *isetl.ini* can be considered to be a (dynamically changing) extension of the language, representing the collective, accumulative, official wisdom of the class.

2. The file *isetl.ini* at this point contains all the group and subgroup functions (is_closed, identity, is_group, is_subgroup, etc.) as well as some standard sets and operations such as Z_{12}, a_{12} (addition mod 12), S_4 (permutations of {1,2,3,4}), os (permutation product), etc. It also contains the func PR, written previously by the students, which implements the "extended product" in a group. PR enables us to form the product of two subsets, or an element and a subset, in any group. The actual code of the func PR appears in Appendix 2.

3. Finally, the file *isetl.ini* contains an additional program called *name_group*, which assigns to a given group (and optionally a subgroup) all the standard notations. For example, assume that Z12 has been assigned in ISETL to denote the set of integers mod 12, a12 the operation of addition mod 12, and H3 the subgroup {0,3,6,9}. If we now execute name_group(Z12,a12,H3) , then the following names are automatically assigned:

G for the set Z12,
o for the operation a12,
e for the identity element 0,
i for the inverse function (so that i(a) is the inverse of a),
oo for the extended product in G,
H for the subgroup H3,
K for the "coset function" (so that K(a) is the coset H .oo a), and
GmodH for the set of all (right) cosets of H in G.

Similarly, if S4, os, and A4 are the symmetric group, the permutation product and the alternating subgroup (all the even permutations), respectively, then executing name_group(S3,os,A3) assigns the same standard names G, o, e, i, oo, H, K, GmodH to the appropriate objects of this system.

Students are encouraged to always use *name_group* in their computer investigations. The purpose is two-fold: First, using standard, short names facilitates the computer work. Secondly (and more profoundly), we believe that using "generic" names, and retaining the same names for different examples, helps students see the example under investigation as being a "generic" example; that is, it helps them in "seeing the general in the particular" (Mason & Pimm, 1984). This, we believe, is a crucial step in the difficult and all-important processes of *generalization and abstraction.*

The actual code for name_group is given in Appendix 3.

Reality --- in the computer lab.

The actual worksheet which the students receive appears in the left column of Figure 2. All of the mathematical expressions that are listed can be written mutatis mutandi in **ISETL** and evaluated on the computer. In the right column are a few remarks about what the students are expected to do and what we expect to be happening in their minds.

The entire activity represents about 60 minutes of cooperative lab work.

(Editor: Put the following Dialogue in a box.)

Figure 2: Worksheet on Lagrange's Theorem

Worksheet	Remarks

H3 := {0,3,6,9};
name_group(Z12,a12,H3);
G; #G; 8 .o 9; is_group(G,o);
e; i(7);
H; is_subgroup(G,o,H);
	Students are asked to predict the result of each expression, then enter it and resolve any conflicts between their predictions and the actual result. The goal for this first group of expressions is for students to refresh their memory about the meaning and interconnections of the many symbols involved in this activity.

H .oo 0; H .oo 1; H .oo 2; H .oo 3; H .oo 4; ... K(0); K(1); K(2); K(3); K(4); ...	Here students are moving on to familiarize themselves with cosets and their various notations. The operation K maps an element of G to its right H-coset. The "three dots" prompt them to look for a pattern.

GmodH; #(GmodH); H subset G; H subset GmodH; H in GmodH; G in GmodH; K(1) in GmodH; K(2) in GmodH; {1,4,7,10} in GmodH; {1,4,7,10} subset GmodH; % union (GmodH) = G; forall a in G : a in K(a);	These are further prompts for exploring the set GmodH of all the right cosets of H in G, with a view to generality (again, using the predict-enter-resolve cycle). The operator "% union" is the extension of the binary operator "union" to operate on any (finite) set of sets.
Find as many equalities as you can among the sets H, K(0),..,K(4), and among their cardinalities. Verify that your equalities return "true".	Computer exploration that could lead to one of the key steps in the proof of Lagrange's theorem.
When is it the case that the relation H .oo a = H .oo b is true?	Students are confronted with another issue in the proof.
Find examples of i /= j in G such that (one condition at a time): (a) K(i) = K(j), (b) K(i) inter K(j) ={}, (c) neither (a) nor (b) holds.	The last major ingredient of the proof. The symbol "/=" is ISETL's way of approximating the mathematical "•" on the standard keyboard.
What is the relation between the numbers #G, #H and #GmodH? Can you see an explanation?	Based on their previous activities and explorations, the students are now ready to try to discover Lagrange's Theorem. Furthermore, since they have also discovered the properties of cosets which are responsible for the truth of the theorem, they now have a pretty good feeling why the theorem is true.
Do name_group(Z12,a12,H4), where H4 = {0,4,8}, and predict the answers to all the previous activities. Check your predictions on the computer. Does the relation you found between #G, #H and #GmodH still hold?	If a student has begun to construct the idea of Lagrange's theorem and its proof, then looking at a second example may help bring it out.

Repeat the same activity with name_group(S3,os,A3).	More of the same with a non-commutative example.

Reality --- in the classroom.

We have seen (Figure 2) students engaged in computer activities, prompted by a worksheet. Based on the experience gained in these activities, students are now further engaged in doing mathematical tasks and calculations in the classroom. Each task is alloted a certain amount of time and the students work on them cooperatively in their teams. While the students are working, the instructor has ample time to move around in the classroom, look at what the students are doing, answer questions and occasionally engage in a dialogue. The activity is then followed by a class discussion. Also, this may be the time for some "official" summaries by the instructor. Some of the outcomes of the class activity are further pursued in homework assignments.

Figure 3 illustrates a typical classroom activity which follows the computer activities in the Lagrange's Theorem worksheet. We give some examples of tasks and describe some typical student responses.

(Editor: Put the following two-column section in a box.)

Figure 3: Tasks on cosets

Tasks	Outcomes
Formulate a conjecture regarding the orders of a finite group G, a subgroup H, and the set of cosets GmodH.	Some teams come up with the conjecture that the number of cosets times the order of the subgroup is equal to the order of the group. Fewer state that the order of the subgroup divides the order of the group. Some students do not see either of these explicitly, but have it on the tip of their tongues, so to speak: they recognize it readily upon hearing it from their peers.
Formulate these points in symbols.	Formulas such as #G = #H * #GmodH o(G) = o(H) * #GmodH #H|#G usually arise.

What are some properties of cosets which seem to "cause" these relations?	The following are easy to observe from the previous activities, and some of them are commonly picked up by the students: - H is one of the cosets. - Every element belongs to some coset. - The union of all the cosets is the whole group. - Different cosets are disjoint. - If a and b belong to the same coset then Ha=Hb. - All the cosets have the same number of elements.

At this point, the instructor may decide that all of the ingredients of the proof of Lagrange's theorem are present in the classroom and, more importantly, have been constructed in the minds of most of the students. The complete formal proof may now be assigned as homework. Alternatively, or additionally, the instructor may tie everything together by presenting the formal statement of Lagrange's theorem and its proof. There's also a third alternative: Following the homework assignment, the instructor discusses with the students their work on the theorem and its proof, and then hands out for them a written page with the complete, polished proof on it (or refers them to the appropriate page in the course text).

Reflections.

As in the first scenario, we now attend to some questions, observations and conjectures as might be posed by our "Idealized Reader".

IR: *The func for PR looks to me like a hard program to write. Doesn't it give students a lot of trouble, and doesn't it involve too much programming effort?*

No on both counts. The main programming ingredient is the "if statement" with all the "elseif" clauses and by the time we get to this activity, the students are quite comfortable with such constructs. Other than this, the program is almost identical with the mathematical definition of this operation.

IR: *What about that line "return func"? I find that pretty confusing. Doesn't it give the students trouble?*

Yes, this is really difficult. This particular func takes as input a set and an operation (which is a function) and returns the generalized operation (which is also a function). One of the hardest things for students to do is to treat functions as total entities, or objects, and perform operations on them. (Sfard, 1992). We see this in many situations, not only in algebra, but also in calculus and other subjects. Here the student expects a func to return a number or even a set of numbers and major mental adjustments are required before the student can understand that the func is to return a "whole function".

But this difficulty is with the mathematics, not the programming. We have done studies which suggest that confronting this kind of mathematical difficulty in a computer context can help students construct the ability to work with functions as objects (Ayres, Davis, Dubinsky, and Lewin, 1986, Breidenbach, Dubinsky, Hawks, and Nichols, 1991).

In fact, it is possible to write an easier version of PR, which takes as inputs the set, the operation and the two objects to be multiplied (an element or a subset), and returns directly their product. We prefer the one given here both because of the opportunity for students to deepen their understanding of functions-as-objects, and because it represents the mathematics involved more faithfully.

IR: *Asking the students (in your worksheet) to characterize the relation "Ha = Hb" seems to me like a regular pencil-and-paper exercise.*

This is true, and if a student prefers to do it this way, that's fine with us. However, many students merely get stuck when facing such a question with only pencil-and-paper as working tools. In the ISETL lab, students still make errors but they rarely get stuck, since they can utilize the computer to analyze examples and conduct investigations. In our experience, most students prefer an environment in which they can utilize such "experimental scaffolding" to aid the theoretical reasoning.

IR: *Do the students really discover all of those coset properties on their own?*

Some do and some don't. It is wonderful for those who do, but again, the point is that this is not necessary. The effort that students put into *trying* to discover these facts is sufficient to change the way in which they respond internally when they hear it from their classmates, or from the instructor.

IR: *Isn't it frustrating for the students who fail to accomplish the given tasks?*

It is very important for the instructor to make sure students don't conceive of this as *failure*. In our classes, we explicitly discuss this issue with the students, and emphasize many times that the point is spending the time and effort on the task, not solving it. In fact, one of our "slogans" (in the similar context of predicting the computer's response) is that, if it turns out that your prediction was correct, then you just wasted your time since you learned nothing new. If, on the other hand, you were surprised by the computer's response then you are lucky, since you now have an opportunity to improve your understanding.

IR: *Well, you have said a lot about alternatives to lecturing, but here, in the end, what you are doing with Lagrange's Theorem is to give a lecture about the theorem and its proof.*

We have no objection to lecturing as such. Our main claim is that introducing new material via a lecture may be a very effective *teaching* method, but it is mostly a very ineffective *learning* method. We use (short!) lectures to summarize and elaborate something the students have previously spent considerable time and effort working on. Ideally, the lecture is on something the students are already familiar with; it only serves to present it in a more explicit, general, precise, formal way.

IR: *The activities have directed the students to some ingredients of the proof, but many remain untouched. For example, they may have discovered that different cosets are disjoint, but there's nothing in the activities to help them actually* prove *this*.

True. In a way we may say that the students now see the proof as being based on a few "main ideas" of the sort you have mentioned, but they haven't yet examined the proof of these main ideas themselves. This "layered" view of the proof is called *a structured proof* and has actually some advantages (Leron ,1983, 1985). In this sense we might say that the student are now familiar with the "top level" of the proof, but they still need to supply the details of the lower levels. In practice, these missing details, are usually assigned as homework.

Third Scenario:
A Gentle Introduction to Normality and Quotient Groups

Background.

The situation at this point in the course is similar to that described at the beginning of the second scenario, and we are making similar use of the file *isetl.ini* and the procedure *name_group* described there. At this point the students have at their disposal, and are pretty comfortable with, the set GmodH of all right cosets of a subgroup H in G, and the "extended product" operation defined on it. This operation had already been extensively used to construct and investigate cosets, but so far we have hardly used the fact that *the same operation can also be used to multiply two cosets*.

Note: We have chosen here a somewhat non-standard route to approaching the notions of normality and the quotient group. In particular, the set GmodH used in the ISETL activities is defined whether H is normal or not. Also, we are initially making an unusual choice for our definition of coset product. Eventually, for a *normal* subgroup H in G, GmodH and this product coincide with the usual G/H and the usual definition of product "by representatives". But with our approach they can be used to great advantage before normality has even been introduced. This is discussed at greater length in the "reflections" section below.

The activity we are about to discuss takes place a little before we are half-time through the course. It typically starts with a short (10-15 minutes) class discussion, in which the previous activities and results concerning subgroups and their cosets are recalled. In particular the following notations are brought into focus (see the second scenario):

K(a) (the same as H .oo a) -- the right coset of *a* relative to a given subgroup H.
GmodH -- the set of right cosets of H in G.
oo -- the extended operation defined on elements and subsets of G.

It is then noted that, since oo enables us to multiply any two cosets, the question naturally arises as to whether (or, better yet, *when*) GmodH is a group under this operation. The students then proceed to do a hands-on investigation, guided by a worksheets as shown in Figures 4,5,6,7.

Reality - Part One: Normality and Quotient group worksheet.

Note: To save room, we give here the comments in between the activities, rather than in the preceding 2-column format. As another space-saving device, we are writing several expressions to a line; in reality the students enter them one by one.

The activities in Figures 4,5,6,7 last *for about 50 minutes. The* students are given instructions which prompt *the*m *to focus their attention on the relevant aspects of a complex situation. One of the pleasant aspects of this kind of non-prescriptive learning environment, is that those who can make sense of such advice will, while those who can't, will just ignore it (at least for the moment), with no harmful side-effects.*

Here are the instructions given to the sutdents.

> In the following activities, predict the answer and then check on the computer. Try to settle any discrepancies that arise.
> Please pay special attention to the following points and write down any observations you may come up with.
> - When is the product of two cosets again a coset?
> - When is (Ha)(Hb) = H(ab)?
> - When is GmodH closed under the operation oo?
> - When is GmodH is a group?

(Editor: Put the following computer output in a box.)

Figure 4: Warm-up Activities

As before, we start with some "warm up" activities, to revive the students' "feel" about the nature of these complicated objects and the relationships between them. ISETL's response is included. Recall that the stuents' input is preceded by a '>' prompt and the computer's output is not. The students' discussion is omitted.

```
>       name_group(Z12,a12,{0,3,6,9});
Group objects defined:  G, o, oo, e, i.
Subgroup objects defined:   H, GmodH, K.
>       G;      H;      #G;      #H;
{0,1,2,3,4,5,6,7,8,9,10.11};   {0,3,6,9};   12;    4;
>       GmodH;     #GmodH;
{{0,3,6,9}, {1,4,7,10}, {2,5,8,11}};   3;
>       H in GmodH;    G in GmodH;    K(1) in GmodH;     K(2) in GmodH;
true;   false;   true;   true;
>       3 in GmodH;    {0,3,6} in GmodH;   {0,3,6,9} in GmodH;
false;   false;   true;
>       {1,4,7,10} in GmodH;    {1,4,7,10} subset GmodH;
true;   false;
>       %union(GmodH) = G;
true;
>       K(0);   K(1);    K(2);    K(3);
{0,3,6,9};    {1,4,7,10};     {2,5,8,11};     {0,3,6,9};
>       K(0) = H;
true;
>       GmodH = {K(a) : a in {0,1,2} };
true;
```

(Editor: Put the following computer output in a box.)

Figure 5: Products of cosets

Following the activities in Figure 4, the students start to look into the product of cosets and its properties. Some of these expressions are saying the same thing, but with different notation and on different levels. It is important (and non-trivial) for

students to be able to experience these different levels of expression as being "the same".
As for the main question, at this point many tend to over-generalize, believing that GmodH is always a group.

```
>       K(1) .oo K(2);
{0,3,6,9};
>       K(1) .oo K(2) in GmodH;
true;
>       K(1) .oo K(2) = K(0);       K(0) .oo K(1) = K(1);       K(2) .oo K(4)=K(6);
true;     true;       false;
>       forall a,b in G | K(a) .oo K(b) =  K(a + b);
false;
>       forall a,b in G | K(a) .oo K(b) =  K(a .o b);
true
>       forall a in G | H .oo a = a .oo H;
true;
>       exists x,y in GmodH | x .oo y notin GmodH;
false'
>       is_closed(GmodH,oo);
true;
>       is_group(GmodH,oo);
true;
>       identity(GmodH,oo);
{0,3,6,9};
```

(Editor: Put the following computer output in a box.)

Figure 6: An example with a non-normal subgroup

A second round through the activities, with a more complex (non-normal) example, gives the students more experience with cosets and the operation on them, as well as a counter-example to a potential misconception. At this point some students decide that GmodH is a group if and only if G is commutative.

```
>       T := {[1,2,3],[2,1,3]};
>       name_group(S3,os,T);
Group objects defined:   G, o, oo, e, i.
Subgroup objects defined:    H,  GmodH,  K .
>       G;        H;
{[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]};  {[1, 2, 3], [2, 1, 3]};
>       #G;       #H;
6;      2;
>       GmodH;       #GmodH;
{{[1, 2, 3], [2, 1, 3]}, {[1, 3, 2], [2, 3, 1]}, {[3, 1, 2], [3, 2, 1]}};              3;
>       p := [3,1,2];              q := [1,3,2];
>       K(p);    #K(p);    K(q);     #K(q);
{[3, 1, 2], [3, 2, 1]};    2;      {[1, 3, 2], [2, 3, 1]};     2;
>       K(p) in GmodH;       K(q) in  GmodH;
```

```
true;      true;
>          K(p) .oo K(q) = K(p .o q);
false;
>          K(p) .oo K(q) in GmodH;
false
>          forall r in G | H .oo r = r .oo H;
false;
>          choose r in G |  H .oo r /=   r .oo H;
[1,3,2];
>          is_group(GmodH,oo);
>          is_closed(GmodH,oo);
false;
false;
>          identity(GmodH,oo);
{[1, 2, 3], [2, 1, 3]};
```

(Editor: Put the following computer output in a box.)

Figure 7: A most complicated case

Coming to the most "mature" example so far (a normal subgroup in a non-commutative group), students start to realize that they need to look deeper into the properties of H that make GmodH a group. They also come to appreciate that the main issue here is closure, namely, when is the product of two cosets again a coset? More specifically, when is it the case that HaHb = Hab? This leads rather smoothly and naturally to the desired property Ha = aH, which is one way of defining normality. It is very important to note that the way they have "constructed" it for themselves, normality is immediately perceived as a way to ensure that GmodH is a group.

```
>          A3 := {[1,2,3],[2,3,1],[3,1,2]};
>          name_group(S3,os,A3);
Group objects defined: G, o, oo, e, i.
Subgroup objects defined:   H,  GmodH,  K .
>          G;         #G;
{[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]};     6;
>          H;        #H;
{[1, 2, 3], [2, 3, 1], [3, 1, 2]};     3;
>          GmodH;           #GmodH;
{{[1, 2, 3], [2, 3, 1], [3, 1, 2]}, {[1, 3, 2], [2, 1, 3], [3, 2, 1]}};      2;
>          forall p,q in G | K(p) .oo K(q) =          K(p .o q);
true;
>          forall p in G | H .oo p = p .oo H;
true;
>          is_group(GmodH,oo);
true;
>          identity(GmodH,oo);
{[1, 2, 3], [2, 3, 1], [3, 1, 2]};
```

Reality - Part Two: Concluding Discussion

This part of the classroom session lasts about 20 minutes, and will not be described here in detail. The instructor leads an interactive discussion in which the different teams share their discoveries, conjectures and questions. By building on the material that the students bring up, the instructor is able to state most naturally and smoothly the definition of a normal subgroup, the theorem that when H is normal then GmodH forms a group, and the (now very easy) proof of this theorem. Normality is naturally introduced here as the condition which insures that GmodH be a group, and the definition most often discovered by the students (while investigating the condition (Ha)(Hb) = H(ab)) is aH = Ha for all a G. Except for the new name, the students can really feel that the instructor merely summarizes what they have found in their investigations. In the session that follows, the instructor makes the final ties with the "standard" approach, by explaining that when H is normal, GmodH is commonly denoted G/H, and is called the *quotient group of G modulo H* and coset product is commonly *defined* by the the formula (Ha)(Hb) = H(ab). In our approach, this formula receives the status of a theorem that comes up in the activities.

Reflections.

IR (Idealized Reader)**:** *In my experience, the quotient group is one place in the course where most students experience a real crisis of meaning. They just stare at the symbols, or hear the instructor's words, and all they see (or hear) is so many ink stains (or so many words). Is there a difference in your class?*

There's no question that this is a difficult concept, and in our class too students have to struggle hard. However, because of the computer activities and the accompanying discussions in the teams, they now have a way of constructing meaning by "successive refinement", as they keep updating and refining their understanding by watching and trying to understand the computer's response to the mathematical expressions they enter.

IR: *What do you mean by "constructing meaning"?*

We believe that this issue is at the heart of students' difficulties in abstract algebra and mathematics in general. Students will often have great difficulty with a theorem and its proof, such as the Homomorphism Theorem or even Lagrange's Theorem. We tend to explain this difficulty by saying that "the theorem is complicated". But is it really the *theorem* that is so complicated? The theorem and its proof are, essentially, about certain relationships which hold between certain mathematical objects. These relationships are fairly simple and we believe that *the difficulty experienced by the students lies not so much in the complexity of the theorem, as in the abstract nature of the mathematical objects involved*.

IR: *Can you give an example?*

Yes, let's look at the relationships involved in Lagrange's theorem and its proof: one number dividing the other, two sets having the same cardinality or being disjoint, etc. These are easy enough. The *objects*, on the other hand, are "complicated" in the sense of the many levels of abstraction and the great time and effort needed to "construct" them in

the mind of the student. For us, the simplicity of the proof lies in our ability to have a clear image of the group as being partitioned into a disjoint union of the cosets. But in order for the students to have such an image, they need to "construct" in their mind not only "group", "subgroup" and "coset" but also "the set of all cosets of H in G".

One powerful aspect of the computer environment we have been describing is that constructing these new entities on the computer, helps students see them as real things that have their own properties, and that can be manipulated, investigated and discussed.

IR: *How can you tell that your method really makes a difference?*

The fact is that students in our classes don't feel the same alienation and paralysis in the face of the quotient group, and end up with a good understanding of it. This is borne out in the research that accompanies our teaching, both in listening to the students' own subjective evaluation of the method and their learning in it, and by in-depth interviews of many individual students. At the very least, they have little difficulty, even on exams, in constructing specific quotients, for example in S4, and identifying the resulting group as being isomorphic to some known groups. Our data suggests that this understanding tends to carry over into ring theory where they can learn to construct and analyze quotients of polynomial rings by various ideals.

IR: *How can you explain the improvement in students' understanding?*

We think it is due to our method which is based on modern understanding of how people learn. This involves a dramatic change in the basic assumptions about how learning occurs: More and more people in our profession are beginning to embrace the notion that knowledge is not *transferred* from one person (the instructor) to another (the student), but rather is *constructed* in the learner's mind. This mental construction takes time and effort and requires an appropriate learning environment the design of which is informed by research into how mathematics is learned

IR: *Exactly how do you use research here?*

The role of research in this context is to propose specific mental constructions that students can make in order to learn specific mathematical concepts, and to propose and evaluate methods for fostering such constructions. For example, to go into a little more detail on a point that was discussed earlier, our research suggests that students' difficulty with understanding Lagrange's theorem may be largely due to their confusion about the nature of cosets. We find that they can understand the process of forming a coset, but often cannot take the next step of seeing these cosets as objects to be measured, counted and compared. Thus, we design computer activities aimed at getting students to construct cosets on the computer and then treat them as objects by manipulating them in various ways. We have done this for a number of mathematical concepts and this work forms a major component of our efforts. We are beginning to develop a body of literature about it which the reader can consult (Ayres et al, 1986; Breidenbach et al, 1991; Dubinsky, 1986, 1989; Dubinsky, Dautermann, Leron and Zazkis, manuscript; Dubinsky, Elterman and Gong, 1989; Dubinsky and Leron, manuscript).

IR: *Isn't it misleading to talk about "GmodH" when H is not a normal subgroup? After all, the product (as usually defined) is not even "well-defined" under these conditions.*

As we have said before, the route to quotient groups as presented here is indeed somewhat unorthodox. We believe that this approach has several learning benefits. Through previous activities (especially of the kind depicted in the Second Scenario), the students have come to feel comfortable with the set GmodH and the operation oo. The question whether GmodH is a group under the operation oo seems most natural here. Our approach allows for a gentle introduction and a smooth, gradual passage to these complicated notions. And the condition for normality emerges naturally through the attempt to answer the question "when is GmodH a group?", or, more specifically, "when is the product of two cosets again a coset?" In the standard approach, the notion of a normal subgroup is usually introduced without sufficient motivation, and the quotient group is introduced rather abruptly, leaving most students no chance to get used to this complicated entity, or as we prefer to put it, to "construct" the appropriate objects in their mind. Add to this the subtleties associated with the definition of coset product by "multiplying representatives", and it is little wonder that they experience a "crisis of meaning".

Summing up, we believe that showing that some familiar set (GmodH) is closed under some familiar operation (oo), is much easier to understand than defining a new set and a new operation and immediately proceeding to prove that the operation is "well-defined" (a notoriously hard concept for students at this stage).

Fourth Scenario:
From the intuitive to the formal -- introducing homomorphisms.

Background.

In this last Scenario we describe a different kind of activity. Here students are presented with an intuitive, intentionally vague "definition" of homomorphism (a "structure-preserving map"), and are asked to write an ISETL func *is_hom* to implement this concept. This is an interesting inversion of the usual routine, in which first a formal definition is given, then an explanation is sometimes added to the effect that this means a "structure preserving map". The ensuing activity is very promising. Students struggle to translate the intuitive idea of a "structure-preserving map" into a formal definition. Once again, they achieve this through a sequence of "successive refinements", aided by the feedback from the computer. When they eventually succeed, it is likely that the formal and the intuitive levels will remain firmly linked in their mind, enabling them to easily shift back and forth between them. In this sense, they'll be functioning in the same way as mathematicians typically do during any problem-solving activity.

The activity takes place shortly after the middle of the term. Just before, there is a summary in class of Lagrange's theorem, normal subgroups and the connection with quotient groups. The computer work on programming is_hom and the kernel of a homomorphism (along with examples) takes 50 minutes and the discussion, culminating in the formal definition of *hom* and *kernel*. takes another 50 minutes.

Usually, at this point, the correct definitions will be offered by a student and a significant percentage of the class will indicate that they also had arrived at such a definition.

Reality, Part I: The Homomorphism Worksheet

Here is what the students read on their worksheet.

1. In your *isetl.ini* file you have the procedure *name_group*, which takes a group (a set and an operation) as input, and assigns standard names to the relevant group objects: G to the set, o to the operation, e to the identity element and i to the inverse function (so that i(a) is the inverse of a)[3]. Since we are now going to work simultaneously with *two* groups, we need to assign standard names also to the second group. Edit the procedure *name_group* to obtain a new procedure *name_group'*, which will assign the names G', o', e' and i' to the corresponding objects in the second group.

2. A homomorphism f : G --> G' is defined (loosely) to be a structure-preserving map, i.e., a function f: G --> G' which preserves the operation. Write a func *is_hom* to implement this concept in ISETL.

3. Can you find a homomorphism from Z_{12} onto Z_4? From Z_{12} onto Z_5? From S_3 onto Z_2? Can you find *two* such homomorphisms?

---

[3] Earlier we have used *name_group* with a subgroup as a third input. This third input is optional, and is not needed here.

Note: The actual worksheet also contains the definition of *kernel* and related activities. Here too the definition is given verbally ("the kernel of f is the set of all elements of G which are mapped to the identity element of G' ") and the students are asked to write it symbolically (as an ISETL func). For reasons of space we omit a detailed description of this part.

Reality, Part II: Students' Work

In Figure 8, we give an example of students' discussion as they are working on the homomorphism worksheet. As before, the left column shows the ISETL code they are writing, and the right column -- the accompanying discussion. Again, for reasons of space we can only include an abridged and streamlined version of a realistic discussion.

(Editor: Place the following in a box.)

Figure 8: Homomorphism Worksheet

	What kind of a func is this is_hom? It has to return true or false... it's a boolean. What are the inputs? Let's see, we have here two groups, G and G', and a function f from G to G'. So we have three inputs: G,G' and f. Wait... we'd better include the operation o as well. Okay, so it's four inputs altogether: G,o,G',f . Now it says here that the function preserves the operation. I am not sure what this means. Let's see... f(a) is a' and f(b) is b' so... how about writing it like this?
> is_hom := func (G,o,G',f); >> return <...> >> end;	Wait a minute! We also need to find a function from Z12 to Z4. Why don't we just divide by 4? Make sure you use div so that you get an integer.
> f := |x-> x div 4|;	Try a few examples.

```
>    f(8);  f(3); f(10);
2;   0;   2;
```

Looks good.
Let's finish is_hom..

```
>    is_hom := func(G,o,G',f);
>>       return
>>          f(a .o b) = a' .o b';
>>                end;
```

```
>   is_hom(Z12,a12,Z4,a4,f);
error: OM + OM
```

Oh, I know, it doesn't know what a and b are. I think we need forall there.

```
>    is_hom := func(G,o,G',f);
>>       return
>>         forall a,b in G :
>>                f(a .o b) = a' .o b';
>>                end;
```

```
>   is_hom(Z12,a12,Z4,a4,f);
error: OM + OM
```

What is it now?
Oh, I bet it's a' and b'... The dumb beast doesn't know we mean f(a) and f(b).
Okay, why don't we just say it!

```
>   is_hom :=
>>   func(G,o,G',f);
>>       return
>>         forall a,b in G :
>>             f(a .o b) = f(a) .o f(b);
>>   end;
```

```
>   is_hom(Z12,a12,Z4,a4,f);
false;
```

Oops! Strange, I thought it ought to be true... Let's try an example.

```
>   f(2 .a12 7) = 2 .a12 1;
false;
```

Let's see... right! 1 is not equal to 3.
Hey, I know. You have to use the other operation on the right.
We want a4 there...
Ok, but that's o'.

```
>   f(2 .a12 7) = 2 .a4 1;
false;
```

Darn! What's wrong now?
You know what? I think our function is no good. What about if we try mod 4?
Ok.

```
>    f := |x-> x mod 4|;
>   f(2 .a12 7) = 2 .a4 3;
true;
```

Try a few examples.

That's it!!
Let's do it over again.
Don't forget, we need to use o', so o' has to be one of the inputs.

```
>   is_hom :=
>>   func(G,o,G',o',f);
>>       return
>>          forall a,b in G :
>>              f(a .o b) = f(a) .o' f(b);
>>   end;
```

Let's try this is_hom with our function.

```
is_hom(Z12,a12,Z4,a4,f);
true;
```

Great!!!

Okay, so our first f was not a homomorphism but this one is.

[*The students now proceed to construct the other examples and the kernel.*]

Reality, Part III: Follow-up -- Examples, Invariants, etc.

In this part the definitions and their meaning are summarized and examples are worked out. More activities concerning the various invariants and non-invariants of homomorphisms are carried out, and the relevant theorems are formulated and proved (largely as a homework assignment).

Reflections.

IR *(Idealized Reader): Some of the mathematical expressions written by your students are even more formal than in the usual abstract algebra classroom. For example, they have to figure out the five inputs to the func* is_hom, *and have to write out explicitly the different operations in the defining relation of homomorphism. In fact the very modelling of the notion of homomorphism as a function (which inputs a function and outputs* true *or* false) *seems*

unusually formal. In my experience, most students find such formalities quite hard and bizarre.

This had been our experience too---as long as we were trying to *present* such formalities in a lecture. We were surprised and delighted to see students arriving at such subtleties on their own, aided by the feedback from the computer and by their on-going discussions. The ISETL medium makes it both necessary and possible for students to deal with such subtleties. The need to "explain" a mathematical concept to a dumb computer is a wonderful motivation for using formal language; watching how the beast actually "understands" (or mis-understands) your explanations, is a wonderful way to climb to the next step in the ladder of successive refinements.

We would like to suggest that there is a distinction between *formalism* and *empty formalism* . When students are asked to deal with a formal statement *before* they have an opportunity to construct the processes and objects that the formality describes, then it is empty formalism. When the formal statement is a description of ideas which already exist in the students mind, then the symbolism becomes a convenient way of communicating these ideas and can even be a powerful tool for further mathematical growth.

IR: *Is it really the case that most students manage to arrive at the definition of homomorphism (as modelled by the ISETL func) on their own?*

This again has to do with the difference between our "constructivist" method and the various discovery methods. All students spend time struggling with the task and experiencing the various constraints and obstacles. Some teams actually arrive at the expected result and some only come close to it.
The main point is that as a result of their computer work, our students tend to form the experiential basis for assimilating the definitions into their existing mental structures when it is finally and "officially" summarized in a class discussion. From this point of view, although those who actually discover the eventual definition may feel happier, the definition will be just about as meaningful to those who have merely spent time *struggling* to discover it.

IR: *At this point, the programming code has become quite involved. Isn't this an added source of difficulty for the students?*

The complexity of the ISETL code faithfully reflects the complexity of the corresponding mathematics. Experience shows that it is much easier for beginning students to manipulate the computational objects (on the computer) than the corresponding mathematical objects (purely in their mind). Similarly, it is easier for them to write code in the "active" programming language, where it can be executed on the computer and the results scrutinized, than to write the corresponding code in the "passive" medium of paper and pencil, where it can only be "run" in their mind. In addition, this scene takes place many weeks into the term, and students by now feel pretty confident about their programming ability. They do encounter many problems on the way to getting their programs to run correctly, but analysis shows that the source of these difficulties is almost always in their incomplete mathematical understanding. The programming medium exposes these weaknesses even as it offers a friendly and efficient environment for gradually eliminating them.

Conclusion

In this paper we have attempted to describe a new paradigm for teaching undergraduate mathematics in general and abstract algebra in particular. It is an expository paper which reports on part of an extensive, on-going research and development project on teaching and learning undergraduate mathematics - activities which have been conducted by the authors and others over the past 10 years. In the project discussed in this paper, we have started out from what we perceive as a general discontent by instructors and students alike with the state of teaching abstract algebra, and have marshaled all the resources at our disposal to develop this new paradigm. These resources include contemporary theory and research into the mental processes involved in learning mathematics, collaborative learning methods, the use of computers equipped with a custom-made programming language, and our own experience over many years of teaching undergraduate mathematics.

We have presented a "constructivist" approach, according to which *telling* students about mathematical processes, objects and relations is not sufficient to induce meaningful learning (hence the sorry state of affairs even with the best of lecturers). What is required, rather, is a learning environment which encourages and enables students to make the necessary *mental constructions* corresponding to these mathematical processes, objects and relations. Forming such learning environments is a highly non-trivial educational task and, in our opinion, the best way of inducing such mental constructions is by having students make appropriate constructions on a computer, and by using the social context to reflect on the computer activities. Our main tool for making constructions on the computer has been programming in ISETL, while our main tool for reflecting on the activities has been collaborative learning, especially discussions among students working in teams. Along with the computer activities and the accompanying discussions in teams, we of course also use the standard tools of class discussion, short summaries by the instructor and homework assignments. However, it is our belief that these tools are most effective when applied *after* the activities.

Making the transition to teaching with this constructivist, interactive method is not easy for the instructor. In addition to making fundamental changes in long-held attitudes about teaching and learning, the instructor must make a substantial initial investment of time and energy for learning techniques very different from the standard lecture method. But our experience and research has shown that this extra effort on the part of the instructor is extremely well rewarded by the change in students' attitudes towards the course and mathematics in general, and by the amount of meaningful learning which is achieved.

References

Ayres, T., G. Davis, E. Dubinsky, and P. Lewin, (1986) *Computer experiences in learning composition of functions*, Journal for Research in Mathematics Education, 19,3, 246-259.

Baxter, N., E. Dubinsky, and G. Levin, (1988) Learning Discrete Mathematics with ISETL, New York: Springer.

Breidenbach, D., E. Dubinsky, J. Hawks, and D. Nichols,

(1991), *Development of the process concept of function*, Educational Studies in Mathematics, 247-285.

Clement, J., J. Lochhead, and E. Soloway, (1980), *Positive effects of computer programming on students' understanding of variables and equations*, Comm. ACM, 467-474

Davis,R.B., C.A. Maher, and N. Noddings, (1990), *Constructivist Views on the Teaching and Learning of Mathematics*, Journal for Research in Mathematics Education, Monograph, No. 4, Reston:NCTM.

Dubinsky, E., (1986), *Teaching mathematical induction I*, The Journal of Mathematical Behavior, 5, 305-317.

Dubinsky, E, (1989), *Teaching mathematical induction II*, The Journal of Mathematical Behavior,8, 285-304.

Dubinsky, E., (in press), *On learning quantification*, in M.S. Arora, (ed.), Mathematics Education: The Present State of the Art, UNESCO, .

Dubinsky, E., J. Dautermann, U. Leron, and R. Zazkis, (manuscript), *On learning fundamental concepts of group theory*.

Dubinsky, E., F. Elterman, and C. Gong, (1989), *The student's construction of quantification*, For the Learning of Mathematics, 8, 2, 44-51.

Dubinsky, E. and U. Leron, (manuscript), Learning Abstract Algebra with ISETL, New York:Springer.

Dubinsky, E. and K. Schwingendorf, (1992), Calculus, Concepts, and Computers: Preliminary version, St. Paul:West.

Leron, U., (1983), *Structuring mathematical proofs*, American Mathematical Monthly, 90, 174-185.

Leron, U., (1985), *Heuristic presentations: The role of structuring*, For the Learning of Mathematics, 5, 3, 7-13.

Mason, J. and D. Pimm, (1984), *Generic examples: Seeing the general in the particular*, Educational Studies in Mathematics, 15(3), 277-289.

Papert, S. (1980), Mindstorms: Children, Computers and powerful ideas, Basic Books.

Selden, A. and J. Selden, (1990), *Constructivism in mathematics education: A view of how people learn*, UME Trends, 2,1, p. 8.

Sfard, A., (1992), *Operational origins of mathematical notions and the quandary of reification -- the case of function.* In G. Harel and E. Dubinsky (Eds.), The Concept

of Function: Aspects of Epistemology and Pedagogy. MAA Notes Series No. 25, Math.
Assn. Amer.

Vinner, S., (1992), *The function concept as a prototype for problems in mathematics learning* In G. Harel and E. Dubinsky (Eds.), The Concept of Function: Aspects of Epistemology and Pedagogy. MAA Notes Series No. 25, Math. Assn. Amer.

Appendix 1. Group axioms as programmed in ISETL by students

```
is_closed := func(G,o);
                return
                  forall a,b in G | a .o b in G;
              end;

is_associative := func(G,o);
                    return
                      forall a,b,c in G | (a .o b) .o c = a .o (b .o c);
                  end;

has_identity := func(G,o);                    $ G has left-identity
                    return
                       exists e in G | (forall a in G | e .o a = a);
                end;

identity := func(G,o);                $ returns the identity element of G
              return
                  choose e in G | (forall a in G | e .o a = a);
            end;

has_inverses := func(G,o);            $  All G's elts have left inverses
                    local e;   e := identity(G,o);
                    return
                        is_defined (e)   and
                        forall a in G | (exists a' in G | a' .o a = e);
                end;

inverse := func(G,o,a);
              local e;   e := identity(G,o);
              return
                 choose a' in G | a' .o a = e;
           end;

 is_group := func(G,o);
                return
                  is_closed(G,o)                          and
                  is_associative(G,o)                     and
                  has_identity(G,o)                       and
                  has_inverses(G,o);
                end;
```

Appendix 2. The func *PR*

In this activity, students are asked to write a func *PR* in ISETL, to represent the concept of the "extended product" in a group. The idea is that, given the original product in any group, we can extend it to operate not only on elements, but on subsets as well. Since PR is to operate on any group, it must accept as input a given group (that is a set and an operation) and return the extended operation on this group. Thus, if Z12, a12 is the additive group of integers mod 12, and if you write

oo := PR(Z12,a12)

then you will have defined oo as the generalized product for that group. For example, here is a short exchange, demonstrating how oo works:

```
>       {0,4,8} .oo 1;
{1,5,9};
>       {1,5,9} .oo {2,6,10};
{3,7,11};
```

In practice, students are given a brief informal explanation of what the extended product means, and are then asked to represent this notion in ISETL. After some initial work, there is usually some class discussion on the type of func that is required here (i,e,. what kind of inputs and outputs), after which the student are left to complete the details of the programming in their teams.

Here is an example of the kind of func students eventually write. (This func is then added to the *isetl.ini* file.)

```
PR :=  func(G,o);
           return
               func(x,y);
                    if  x in G and y in G then return
                        x .o y;
                    elseif x in G and y subset G then return
                        { x .o b : b in y };
                    elseif x subset G and y in G then return
                    { a .o y : a in x };
                    elseif x subset G and y subset G then return
                        { a .o b : a in x, b in y};
                    end if;
               end ;
       end ;
```

Appendix 3. The func *name_group*

```
name_group :=        proc(set,operation opt subst);
                         G := set;   o := operation;    oo := PR(G,o);
                         e := identity(G,o);  i :=  |g -> inverse(G,o,g)|;
                         writeln "Group objects defined:     G,  o,  oo, e,  i .";
                         if is_defined(subst) then
                             H := subst;
                             GmodH := {H .oo g : g in G};    $ The sets of right cosets
```

```
        K := | g -> H .oo g |;          $  K(g) is the coset of g
        writeln "Subgroup objects defined:  H,  GmodH,  K .";
    end;
end;
```