30

Computer integration in the mathematics study of pre-service teacher education - experiences with Project CIMS

Herbert M. Loethe
University of Education Ludwigsburg
Postfach, D-71602 Ludwigsburg, Germany

Abstract

A compulsory course for beginners in 'Introduction to Mathematics' integrates constructive and algorithmic aspects of the concepts involved by using the functional-applicative language Scheme with subject specific and notational extensions (Scheme-L). This course is a basis for further courses in mathematics and computer science.

Keywords

Elementary education, secondary education, teacher education, algorithms, mathematics, teaching materials

1 INTRODUCTION

There are currently two areas of computer innovation in German schools: the use of computers as an activity in different subjects (mainly with standard software, called 'basic education in information technology') and computer science as a separate subject (called 'informatics') or as units of school mathematics. In most of the German states the basic education in information technology is a compulsory part of the instruction at a secondary school level. The teaching of informatics is mostly optional. Teachers of mathematics play an important role in teaching both aspects and in organising and managing the hard- and software installations in schools.

Three observations emerging after more than 20 years of innovative work in this field led us to the starting theses of our project CIMS. Firstly, in-service training is in general limited and unable to create fundamental knowledge and habits. Consequently, in-service trained teachers in general do not change their practice and are slow in picking up innovations in the field of computer use and computer science. Basic education in the use of computers should be a part of pre-service teacher education.

Secondly, even the use of computers with standard software in school requires teachers to have a fundamental knowledge about computers and the basic concepts of computer science. It is not sufficient for effective teaching that teachers are only one step ahead of their students in managing the software. They are undertaking major changes in their subject without any background knowledge and without being able to aim at certain objectives. The teacher has to make sure that such activities in school contribute to the education of the students and are not just a training in software use.

Thirdly, teachers of mathematics are the ones who have to be educated to establish some kind of computer literacy in school which is not restricted to the mere use of standard software. Unfortunately, 'computer literacy' has been redefined every five years in the past. Our thesis is that the fundamental concepts of computer science should be taught, closely related to school mathematics, since the didactics of mathematics (as a science) has the methods and experience to deal with the goals of general education.

2 MATHEMATICS AND INFORMATICS IN PROJECT CIMS

If we consider computer science or informatics more as a scientific and less of an engineering discipline we can see that it is similar to mathematics in content, thinking style and working habit; informatics deals with the 'dynamic' aspects of the abstract representation of knowledge, while mathematics stresses the 'static' ones. In addition to that, some concepts and contents of computer science are so important and fundamental that they must enrich future school mathematics. The research discipline of mathematics cannot any longer be the only reference for defining the school subject.

It is our strong feeling that traditional school mathematics has to change. On the one hand, the applications of mathematical methods are mostly applied in the reality of science, industry, and commerce when using computers. On the other, larger parts of the school mathematics curriculum should be rethought and redesigned because of the impact of Computer Algebra Systems (CAS) which trivialise larger parts of the subject.

We can compare the new role of informatics in relation to school mathematics to geometry. Geometry has grown out of geodesy (the engineering aspect) and the application of mathematical methods has not been driven from the subject (like theoretical physics). There was always an intensive interaction between abstract mathematics and geometry, as illustrative application. Mathematical structures, thinking styles, representations, etc. were induced by geometry. In a comparable way, some sub-disciplines of informatics should be incorporated into school mathematics as constructive applications. A concrete algorithmic style of thinking should be a training in school and used to work out specific aspects and facets of mathematics. In our present judgement the functional-applicative style of thinking and writing algorithms is an adequate one for school and teacher education.

As a consequence of our perspective of the situation and future development, we started the project CIMS (Computer Integration in the Mathematics Study) for reorganising the mathematics study of teachers at our university. Since 1991 every student teacher for secondary level mathematics (and, since 1993, in addition, for elementary level mathematics) has to work through an introductory course

'Introduction to Mathematics' (logic, set theory, combinations, concepts of numbers, theory of numbers, mathematical structures). The important point is that after years of optional courses in computer science for student teachers the department came to the opinion that all students of mathematics have to do computer work at this level of abstraction at the outset of their study (using a local network and text processing systems goes without saying). As a conceptual bridge between mathematics and computer science we referred to the concept of 'function' and the applicative paradigm of programming.

Traditionally the course 'Introduction to Mathematics' was taught in lectures with smaller practise groups doing related paper and pencil work. We enrich the course in several ways. The lecture is done in the traditional style, but it integrates mathematics with dynamic concepts of informatics. It is now based on an elaborated textbook. The left hand pages contain just text in a rather concentrated form: definitions, theorems, proofs, concepts of computer representations, algorithms, etc. This part is not presented for reading in advance or for self-study. The lecturer will teach the subject matter and the students do not need to make notes. During the teaching process the lecturer will sketch illustrations, calculate examples, perform algorithms, assign small exercises and so on. All these activities are prepared on the right hand pages of the textbook and can be completed by the students. After certain sections, traditional paper-and-pencil exercises are assigned as material for tutored group work. In the same way, the practical work with computers is included in the textbook. But not only are assignments given, there are also explanations of language elements and programming techniques, etc. Students usually work through this material in tutored computer sessions in the computer laboratory, but the material is written in a way that they can also work at any other time, and elsewhere, using our network or their home PC. The intended working style of students in the computer sessions can be characterised as a learning by interaction with the dialogue system: experimenting and exploring, learning by debugging, self-demonstration of concepts, and so on. In the context of CAI, this style of computer use for learning was often called the problem solving mode, and it is part of the LOGO philosophy (for example, Papert, 1980). Like Papert, we feel that giving a lecture about mathematics is not sufficient, students have to be put in a situation for actively undertaking mathematics.

3 THE INTERACTION BETWEEN LECTURE AND COMPUTER WORK

On the basis of PC Scheme (TI, 1990) we have programmed extensions which are closely related to the subject matter of the course. There are just translations of keywords into German (since German wording is often more precise than English); the use of infix-notation for algebraic operators and others, many special procedures and functions for mathematics, etc. For instance, suitable co-ordinate graphics, turtle and space turtle graphics packages (Loethe, 1992) are implemented. Some of this work is comparable with Simply Scheme of Harvey-Wright (1994), but our packages are specifically designed for use in the context of mathematics.

The students are introduced to the Scheme-L System by means of turtle graphics. After this, they use the system as a calculator for factorial and binomial coefficients, while in the lecture the theory of combinations are taught. In addition, they use the

system as a medium for self-demonstrating the different combinations by using pre-defined functions. This mode of self-demonstration is considered an important way of working with the mathematical concepts concretely, trying out conjectures, experimenting, getting a feeling, for example, for the increasing number of combinations, and so on. In the context of the theory of combinations the mathematical concept of 'set' and the informatical concept of 'list' are introduced as complementary techniques of data abstraction. Lists are used throughout the course as the fundamental data structure in Scheme.

The conceptual interaction between mathematics and informatics should be made clear by the discussion of the concept 'function' to some extent, which is fundamental for mathematics and functional-applicative languages like Scheme. Every kind of mathematical object can be transformed by a function; there are no restrictions in mathematics. This fact excludes imperative computer languages (with the von-Neumann bottle neck) to be considered as reasonable tools for computer use in mathematics. The concept of a pure function is exactly the same in Scheme and in mathematics (besides the fact that in mathematics infinite domains are allowed also). In Scheme-L we can use the arrow notation as an infix-notation which is closer to mathematics than the prefix lambda-notation of standard Scheme. The following is the functional object 'squaring':

```
((x) -> (x * x))
```

The naming of a functional object is done by one of the following expressions:

```
(square:=((x)->(x*x)))
(def square((x)->(x*x)))
```

An alternative way of defining a function is to use the calling pattern:

```
(def (square x) (x * x))

[1](square 3)
9
```

This flexibility in notation is an important aspect of the methodological design of the course. The system has to be accommodated by the learner and the learner should not be stressed in having to deal with strange notations if this is not necessary. Another case is the use of parentheses. Since the standard mathematical notations are so ambiguous, the Lisp notation is unavoidable.

The variable x in these expressions is a so-called bound variable. The concept of bound versus free variables is important for both mathematics and computer science:

```
(def (parabola x) (a * x * x))
```

Traditional mathematics does not make this concept very clear, and refers to a more psychological than logical way of thinking about them, such as: free variables are called 'parameters' or a variable which varies 'slower' than a bound variable; or a free variable is something which has a value in a certain working context. Computer

languages have a specific technique (called scoping) for looking for the value of the free variable in certain environments.

Another aspect of the concept of function is that functions can have side effects on the environment, the screen or other devices. Traditional mathematics does not deal well with this aspect (except for the concept of automata). Here we have a possible enrichment of the traditional mathematical concept of 'function'. In our course functions can have side effects on the environment (database or memory). By incorporating this typical technique of imperative programming into the concept of function we can establish a link to imperative programming without stressing it. In general we avoid the imperative thinking style about algorithms and prefer the recursive one.

Recursively formulated algorithms are thought to be abstract and difficult to understand. In the context of our course, we do not have these difficulties since the problems are rather small and mostly last-line recursive. In order to give an example, consider the divisor list after a test number t of an integer n:

```
(def (divisor-list-after t n)
  (cond ((t > n) ())
        ((t divides n)
         (cons t (divisor-list-after (t + 1) n)))
        (else  (divisor-list-after (t + 1) n))))
```

('divides' is a pre-defined infix-operator for number theory.) The students need only to understand the function which is presented and explained in the lecture, to change it to a more efficient version, and to write functions with an analogue structure.

To learn a mathematical theorem means, in our opinion, it is necessary not only to understand its content, the domain of applicability and the idea of its proof, but also to know its constructive use, for example, as an algorithm. For instance, the theorem of Euclid states that there are infinitely many prime numbers. This is closely related to the function 'the next prime number after a given number':

```
(def (next-prime-after n)
  (if (prime? (n + 1))
      (n+1)
      (next-prime-after (n + 1))))
```

(prime? is a predicate which was developed previously in the lecture and tested in the practical computer work by the students.) The constructive meaning of the Euclid theorem is that his function always stops since the 'next' prime number always exists. The proof cannot be given by arguments related to the finite algorithm, but the mathematical proof of the theorem is strongly motivated by this constructive aspect.

It is by no means the goal of this course to give an introduction to programming with Scheme. The examples used are rather simple, and closely related to mathematical problems. We are more concerned with the concepts and their constructive aspects. The example divisor-list-after offered describes the upper limit of complexity of a Scheme function in this introductory course.

4 EXPERIENCES

The course is still an innovation in process. Evaluations in the past by questionnaire or tutor comments were used to redesign the concept and rewrite the material over a one year cycle. Some of the results of our last questionnaire gathered after the winter semester course 1995/96, indicated that, in general, the students very much liked the elaborated manuscript, especially the structuring of the theory (left pages) and activities (right pages): 73% indicated that it was helpful, 21% neutral, 3% not necessary, and the rest were at the extreme ends. The statement "The right pages encourage me to work actively and struggle with the subject matter" was commented upon: 4.5% indicated that it was not true, 23.9% neutral, 64.2% true, and 7.5% that it was strongly true. The statements about the interaction between mathematics and computer work with Scheme are the most important for our project. We report them in more detail. The students were asked to comment on the statements using a 5 point scale:

not at all true --- not true --- neutral --- true --- strongly true.

We give the percentage of their answers (n=67 in most cases) in parentheses: "Mathematics education should be done without computers, since undertaking both together is too difficult" (9.0%, 35.8%, 22.4%, 25.4%, 6.0%); "I got a better understanding of mathematical concepts and methods by undertaking mathematical and computer work in an integrated way" (9.0%, 22.4%, 28.4%, 26.9%, 11.9%).

While these comments are rather positive, the concrete work with Scheme was reported more negatively: "The work and challenge in using Scheme supports my mathematical understanding" (20.9%, 31.3%, 32.8%, 10.4%, 4.5%); "The work with Scheme demands a high level of abstraction" (1.5%, 6.0%, 20.9%, 55.2%, 14.9%); "The work with Scheme demonstrates mathematical concepts to me" (10.4%, 22.4%, 40.3%, 25.5%, 1.5%). However, some responses do not seem to be very serious: "Dealing with Scheme disguises the mathematical content" (9.0%, 32.8%, 31.3%, 20.9%, 6.0%); "I do not understand recursive programs" (20.9%, 31.3%, 26.9%, 10.4%, 6.0%). The reaction to the statement: "The parentheses in Scheme are extremely troublesome" (7.5%, 11.9%, 11.4%, 20.9%, 40.3%) is considered to be emotionally derived due to the difficulty the beginners had in computer sessions. Later on in their study, students realised the merits of parentheses and complained in other courses with computer activities about the lack of transparent syntax in Logo or Maple.

The questionnaire was given at the end of the course. All students who did not complete the course are not represented. Unfortunately, we have at the moment no control over the number of these students, or over their motivation and plans. They can change the subject of their study or postpone the course. The results given seem to be rather too positive in comparison to our other observations (but in any case are very encouraging for the project). One important general observation of lecturers and tutors is that the whole student group becomes polarised by the practical computer work. There are students who are attracted to the computer work and get a lot of positive feedback on their work in mathematics. On the other hand there are students who try to avoid computer work completely. An extensive study on the individual working style of students in the whole learning environment is presently under way. The most encouraging result is the unbiased and natural way that students use

computers later in other courses of their study. So, we are able to observe a process of habituation to this integrated work in the whole student population.

I would like to thank Rose Vogel for her intensive work in the project and all other colleagues of the department for their contributions and encouragement.

5 REFERENCES

Harvey, B. and Wright, M. *Simply Scheme: Introducing Computer Science.* The MIT Press, Cambridge Massachusetts, London, England.

Loethe, H. (1992) Conceptually Defined Turtles, in C. Hoyles and R. Noss (eds.) *Learning Mathematics and Logo,* pp. 55-95. The MIT Press, Cambridge Massachusetts, London, England.

Papert, S. (1980) *Mindstorms: Children, computers, and powerful ideas.* Basic Books, New York.

TI (1990) *PC Scheme: User's Guide and Language Reference Manual.* The MIT Press, Cambridge, Massachusetts, London, England.

6 BIOGRAPHY

Herbert Loethe is a professor of mathematics and computer science at the University of Education Ludwigsburg, Germany. He has done extensive work on Computer Assisted Instruction and Computer Sciences in schools since the early 1970s. More recently he has worked on the introduction of Logo languages in schools and teacher training, and its relationship to mathematics.