The Role of Discrete Mathematics and Programming in Education

Sylvia da Rosa

Instituto de Computación - Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay
darosa@fing.edu.uy

Abstract. We have been teaching introductory courses on Functional Programming, as part of the curricula of computer science University studies, for several years. We claim that the origin of most of the difficulties in learning Functional Programming doesn't lie in computer science courses but in the mathematical background of the students. We are convinced that an integrated work of mathematics and computer science educators will considerably benefit the learning of both subjects. Our proposal mainly consists in formulating a new approach to teach discrete mathematics topics included in High School mathematics curricula, relating them with computer science concepts. This implies to increase the dedication to discrete mathematics in High School curricula and to introduce a programming language as a formalism to manipulate the mathematical objects. We include examples of a course in Discrete Mathematics using the programming language ISETL that we taught to a group of High School mathematics teachers.

1 Introduction

We have detected that the students entering the University, have serious difficulties to understand concepts such as high order functions, recursion, induction, type systems and logic in general. On the other hand, those concepts are widely *used* by the students of High School, to acquire *calculation skills*: they work with derivatives, (high order functions), they use sets as domains and range of functions (similar to type systems, to some extent), they use recursive algorithms (for instance, Euclide's Algorithm to find the greatest common divisor of two given numbers) and they make proofs using mathematical induction (induction and logic).

We completely agree with the thought that Declarative Programming is a wonderful vehicle to introduce computer science concepts and notions, which implies that improving the learning in this area is a general contribution to the education in the field. For the same reasons, the difficulties that the student s fail to surmount, act as a barrier to success in future studies.

However, the difficulties in learning the concepts are not exclusive of computer science education, on the contrary, they are also pointed out by mathematics educators [4] [3]. Of course, this means that the problem is much harder to solve,

because it is concerned with changes involving not only computer science education but also mathematics education.
Several efforts are made to improve the learning process: from mathematics' side, for instance, various software systems are introduced in mathematics courses as much to illustrate ideas, and concepts, as to produce efficient results (calculations, graphs, 3D pictures,etc) and from computer science's side, the concern about educational questions in the field has increased significantly [2] [3].
Although we think that these efforts are really useful and positive, we claim that they are not sufficient; they are isolated attempts to solve a problem that is common both for mathematics and computer science education. Neither using software or programming languages in mathematics courses merely as a tool nor ignoring the defficiencies of the mathematical background of the students of computer science, will help in the effective solution to the problem. On the contrary, what is needed, in our opinion, is the assembled work both of mathematics and computer science educators to create and teach new courses to be introduced as part of the basic mathematical knowledge required to pursue scientific or technical studies. In those courses, discrete mathematics must play a central role and even more important, the relationship between it and computer science must be explicitly stressed.
Almost all the time, High School mathematics courses are dedicated to studying Calculus. In this way, the students believe that mathematics is always continous and that all is related to the field of real numbers. We surveyed some groups of students at the first computer science year and we found some results that confirm this fact, for instance, they never describe a function including domain and range (all is R), they hardly ever use either quantifiers carefully in proofs or notions of logic in calculations (what counts are the results of calculations), they treated induction as a recipe to solve certain problems (often involving sums of numbers).
In our approach we demonstrate how much can be done *without* the field of real numbers, to struggle against the idea that mathematics is always continous. We put our proposal into practice, teaching a discrete mathematics course using a programming language (ISETL) to a group of High School mathematics teachers. As final work, they were demanded to apply our ideas in their High School courses, introducing a selected topic according to our approach and using ISETL. In section 3, we describe an example extracted from the work of some of the teachers with a class of sixteen year-old students.[1]

2 The approach

The main goals of our activity are:

- to explain the relationship between computer science and discrete mathematics through concepts common to both subjects, such as function, algorithm, inductive definitions, etc,

[1] In other countries, this proposal is perhaps more appropiate to College or University courses.

– to show the benefits of including a programming language, for instance: it demands to be careful in writing definitions and it makes possible "to forget" the calculations, allowing the development of other skills such as the process of solving a problem thinking about what must be done and not how to do it,
– to stress the process of abstraction through the construction, step by step, of problem solutions from their specifications,
– to remark the importance of logic to understand what definitions, properties, proofs, etc, are,
– to demonstrate that computer science involves a lot of interesting mathematical activities.

Since our students (High School mathematics teachers) have no knowledge of programming, we use ISETL, a programming language especially designed to be used in mathematics courses and thus advantageous to our task. Its syntax is very similar to mathematics (for example, sets are written as in mathematics and their operations are predefined). ISETL is not a typed functional language (which are, in our opinion, the most desired features) but it allows the definition of functions in a λ calculus manner, including high order and recursive ones. We use it as a functional language, that is to say, we program in ISETL just mathematical objects, so we do not have to explain what a while is, for instance.
Two of the topics of our course deserve special attention and discussion: the concepts of function and induction. In the following, we summarize the main debated arguments about them because they clearly illustrate the difference between some aspects of our approach and the traditional one.
Mathematics teachers amply recognize the fact that most of the students are restricted in their understanding of function to the idea of an algebraic formula for computing values, and that without it they are, in general, not willing to agree that a function is present. Hence the difficulty in dealing with functions as objects that can be operated on, applied to functions, etc. This uncomplete notion of function has undesirable effects in learning as much mathematical as computer science concepts and unfortunately it is strongly rooted in the minds of the students. However, a function is usually introduced as a special relation, that is to say, as a subset of a cartesian product of two sets. In association with this definition, a representation using arrows between points (elements in sets, domain and range) is often presented. This introductory notion of function is very close to the idea of a process: the arrows represent a "transformation" or "rule of correspondence" and the domain and range are present, even more they can be any sets, including not numerical ones. No algebraic formula appears. In fact, the concept of function presented to the students first, is very close to the notion of process, method, algorithm, represented by the arrows. In this way, concepts like compositions of functions, high order functions (functions as objects), functions from any set to any set, would be nothing else than naturally derived concepts.
But this natural way of development of the concept of function and concepts derived from it, quickly deviates towards the study of a *particular case* of functions:

the functions whose domain and range are subsets of the set of real numbers and an algebraic formula computing its values exists. This particular case belongs to Calculus courses and because of the excessive weight of those courses in the curricula, the students "forget" the original concept of function.
The other point is the concept of methods of proof in general and induction in particular and their relation with logic, which is rarely taken into account. Making proofs by induction the students think on induction as a "technique" or a recipe to prove some properties usualy involving sums. In our course, we presented to the teachers, the concept of induction related with :

1. the inductive definition of sets (inductive sets),
2. the definition of a function whose domain is an inductive set, that is to say, using equations and patterns,
3. the proof of properties on inductive sets using the corresponding induction principle.

Some exercises using structural induction to prove properties about several inductive sets are presented, for instance the set of wff of propositional calculus.
The adopted methodology is similar to the one described in [1], in the sense that the students work on several activities before the formal treatment of the concepts. Through the activities the students can make their own discoveries. On the other hand our methodology differs from the cited one, in the sense that we always demanded to define the concepts mathematically first and then to translate the definition to an ISETL program. We think that in that way, the relationship between mathematics and computer science is clearly stated.

3 Practical examples

As final work of the course, the teachers applied the discussed ideas, the described methodology and ISETL, to introduce in their High School classes, one of the topics of the course, according to our approach. We present here a summary of one of the works, held in a class of 13 students age 16, in which the selected topic includes division in Z, multiples, divisors, Euclide's Algorithm to find the greatest common divisor of two numbers. The teachers' plan of classes described here, includes, for each class, the pursued goals, the activities and their comments, which we have marked "T. comments", in order to distinguish them from our own comments, marked "Our comments". Mathematical definitions are written in italic type.

First class:
Goals: To arrive at the definition of division in Z and analyze the fact that the requisite of the remainder being greater than zero, guarantees that the remainder is unique.

Activity 1:
copy the following code and run it:

```
> for x in [10 .. 25] do
```

```
>>        writeln x, x div 6, x mod 6
>> end;
```

What do *mod* and *div* "do" ?

Activity 2:
a) repeat activity 1 in the case x in [-10 .. 10].
b) repeat activity 1 in the case x div 8, x mod 8.
c) repeat activity 1 in the case x in [-10 .. 10] and x div 8, x mod 8.
d) write your own observations.

T. Comments: We consider that ISETL allows the students to pay attention to the results and not to the operations. For example, the students were able to define the operation of Division in Z: $\forall$a,b $\in$ Z, $\exists$ c $\in$ Z and r | a=b*c + r with 0 < r < b, and to observe which are the possible remainders according to the divisor. They observed that the condition of r > 0 guarantees that the results are unique, as well. The proof is made in the usual way.

Our comments: observe that the teachers' comments match up to one of our goals ("forgetting the calculations"). Observe that usually, the operations div and mod are not used in High School mathematics.

Second class:

Goals: To obtain the definitions of the set of multiples of a number, both mathematically and in ISETL. Through abstraction, arrive at the definition of a function that given a number returns the set of its multiples.

Previous activities (not included here for space reasons), allowed the students to arrive at the following definitions: a is a multiple of b if and only if a mod b = 0. The definition of sets using set formers and the concept of function had been introduced before, mathematically and in ISETL.

Activity 3:
a) using the definition above, define the set of multiples of 12 between 30 and 100 as a set former.
b) write the definition in ISETL.
c) set a suitable name to this set.

T. Comments: The aim of part a) and b) is that the students arrive at both the mathematical definition:
$\{x \mid x \in [30, 31, \ldots 100] \wedge x \; mod \; 12 = 0\}$
and the ISETL code:
{x : x in [30 .. 100] | x mod 12 = 0}
In part c), they are guided to name the set with an expression containing 12, in order to substitute other numbers for 12 and arrive at the definition of a function. In doing so, the students are reminded about a similar case: the name of the power set of a set A, P(A), in which P is a function, indeed.
The result of part c) is, thus:
$mult(12)=\{x \mid x \in [30, 31, \ldots 100] \wedge x \; mod \; 12=0\}$
mul12 := {x : x in [30 .. 100] | x mod 12 = 0}

Our comments: The comparison between ISETL's formalism and mathematics one brought out interesting questions to be discussed, for instance: What hap-

pens if we omit the constraint "x in [30 .. 100]"? Can ISETL compute all the multiples of 12 ? Why does ISETL use the := symbol instead of = like mathematics ?
The first two questions provide the opportunity for discussing about the limitations of the computer.
The last question touchs an important concept, in our opinion: in mathematics, the use of the same symbol = with different meanings unconsciously confuses the students while a programming language avoids this type of confusion and provides the opportunity for discussing about languages' semantics.

Activity 4:
a) apply the ISETL definition above, in order to find the set of multiples between 30 and 100 of several numbers, for instance 20, 10, 35, 55, 110. (Actually, this activity asks for filling a two-columns table one of them containing the number and the other the set of multiples).
b) is the stated correspondence a function ? If yes, which are its domain and range ?
c) define the function both mathematically and in ISETL.
T. Comments: Taking into account part c) of the previous activity, the students easily arrived at a definition of the function and an ISETL program:
$mul : N \rightarrow P(N)$
$mul(a)=\{x \mid x \in [30, 31, \ldots 100] \wedge x \; mod \; a=0\}$

```
mul:=func(a);
     if is_nat(a) then
     return {x:x in [30 .. 100] | x mod a=0};
end;
```

Our comments: A question deserving special attention arises here: although ISETL is, in our opinion, a well suited programming language to be used for the purpose of this work, it lacks some fundamental features: for instance, it doesn't demand the inclusion of the signature of the function being part of its definition. In the case of, say, the following ISETL definition:

```
f := func(x);
      return 2 * x;
end;
```

f defines all the functions over all the sets for which the operation (2 *) is defined. Observe that the definition of the function mul above, includes the clause "if is_nat(a) then ...", which defines the function : N $\rightarrow$ P(N), as required. If the argument is not a natural number, the program returns OM, which is ISETL's denotation to the undefined element, indicating that the function is not defined for values $\notin$ N. In our course we always strongly demanded both the signature of the function in its mathematical definition and the inclusion of the if clause in the ISETL definition. This is not only mathematically correct, but also allows to verify some function's properties such as if it is total or partial, if it is injective, etc.

Third class:
Goals: To introduce the concept of least common multiple. To construct definitions from specifications.

Activity 5:
define a function that given a natural number n, returns the set of *positive multiples of n* less than a given bound.
T. Comments: In the same way that before, the students arrive at definitions like:
$mul : N \times N \rightarrow P(N)$
$mul\ (a,bound) = \{x \mid x \in N \wedge x \leq bound \wedge x\ mod\ a=0\}$

```
mul:=func(a,bound);
     if is_nat(a) and is_pos(bound) then
     return {x:x in [1 .. bound] | x mod a=0};
end;
```

Our comments: Defining functions in this way is not usual in High School. The students learn the concept of multiples of a number as being a set and the function that computes the algorithm to obtain it hardly ever is treated. We think that these examples clearly illustrate what we mean with an approach that integrates discrete mathematics and computer science.

Activity 6:
a) define the set of *common multiples* of two given natural numbers a and b.
b) state the specification of *the least common multiple* of two given natural numbers in natural language (English, Spanish, etc.)
T. Comments: The aim is that the students begin with the specification of common multiples in natural language and arrive at definitions of both a function and a program, to obtain the least of them, step by step and through abstraction. Using the function mul defined early and their specification in natural language (not included here), they can express the set of common multiples of two given numbers a and b as A = mul(a) $\cap$ mul(b). Computing this set in ISETL, demands a bound and the product a * b suffices: A:= mul(a,a * b) inter mul(b, a * b), where inter is $\cap$ in ISETL.
Our comments: According to the specification of least common multiple, it was required to obtain the minimun element of the set A. In order to do so, we provided a function XR (not included here), that given a set X and an order relation R, returns the "minimun" element of X according R. This means that we found it difficult for the teachers to program such a function and we added it as a predefined one. The environment of predefined functions to be provided must be determined in each case, according to the specific course. Here appears once again the need and the benefits of the collaboration between mathematics educators and computer science ones. We also defined "less", to be used as argument to XR:

```
less:= func(x);
      if is_nat(x) then
```

```
      return |y -> x < y|;
end;
```

Although these functions were provided, their definitions were discussed in the course, introducing the notion of high order function definition, like less.
T. Comments: The students were capable of using ISETL to compute *the lcm of several numbers a, b*, for instance:
> bound := 360 * 405;
> A := mul (360,bound) inter mul(405,bound);
> XR (A,less);
> 3240;
and consequentely, *abstracting the numbers a, b to define a lcm function:*

```
lcm:= func(a,b);
      local bound; A;
      if is_nat(a) and is_nat(b) then
      bound:=a * b;
      A:= mul(a,bound) inter mul(b,bound);
      return XR(A,less);
      end;
end;
> lcm(360,405);
> 3240;
```

All the functions were defined mathemathically as well, however, we do not include the definitions here for space reasons. The discussion of the advantages/disadvantages of including the if clause in the lcm function, (observe that mul checks it), provides the opportunity for introducing notions as algorithms' efficiency, computers' resourses, clarity and number of lines of a program.

Fourth class:
Goals: To introduce the concept of divisors of a number. To find an algorithm to obtain the set of divisors of a given number and define a function that computes it. To define prime numbers.
T. Comments: Through several activities similar to the previous ones, the students arrive at the following definitions[2]:
divisors : $N \rightarrow P(N)$
divisors $(n) = \{x \mid x \in N \wedge x \leq n \wedge n \text{ mod } x = 0\}$

```
divisors:= func(n);
           if is_nat(n) then;
           return {x:x in [1 .. n] | n mod x=0};
end;
```

$$prime : N \rightarrow Bool$$

$$prime(n) = \begin{cases} false & ; \quad if\, n = 1 \\ divisors(n) = \{1, n\} & ; \quad otherwise \end{cases}$$

[2] The set Bool was defined as {true, false} in propositional logic.

```
prime:=func(n);
        if is_nat(n) then
           if n = 1 then return false
           else return (divisors(n) = {1,n});
           end;
end;
```

Fifth class:
Goals: To introduce Euclide's Algorithm to find the gcd of two numbers. To introduce the concept of recursive definition. To state what a property and its proof are.
T. Comments: Euclide's Algorithm is presented as usual and various activities are proposed in order to let the students experiment with different cases to get the idea of how the algorithm works. They are asked to specify it in natural language and step by step, arrive at a definition of a function that computes it:

$$gcd : Z \times Z \rightarrow Z$$

$$gcd(a,b) = \begin{cases} b & ; \quad if\ a \bmod b = 0 \\ gcd(b, a \bmod b) & ; \quad otherwise \end{cases}$$

and the corresponding ISETL program:

```
gcd:= func(a,b);
        local r;
        if is_integer(a) and is_integer(b)
        then r := a mod b;
             if r = 0 then return b;
             else return gcd(b,r);
        end;
end;
```

T. Comments: The concept of recursion is discussed here, recalling the importance of the base case and the application of the function gcd to, in some sense, "less" arguments, in order to reach the base case and guarantee the termination of the program.

Activity 7:
using a function max (also provided), that given a finite set of numbers, returns the greatest of them, and the function divisors previously defined, write another definition of the function gcd.
T. Comments: The aim is that the students arrive at:
gcd1: $Z \times Z \rightarrow Z$
gcd1 (a,b) = max (divisors(a) $\cap$ *divisors(b))*
Having two definitions of the function that computes the greatest common divisor of two integers, it naturally arises the need of stating the equivalence between them, that is to say, the proof of the property:
$\forall$ a, b $\in$ Z, gcd (a,b) = gcd1 (a,b), that is to say,
$\forall$ a, b $\in$ Z, gcd (a,b) = max (divisors(a) $\cap$ divisors(b)).

4 Conclusions and Further Work

The evaluation of the experiment, partially described in this paper through the examples, revealed encouraging results. We are improving and working on various aspects of our proposal, being one of them, the implementation of a programming language more suited to teach discrete mathematics in High School.

References

1. Dubinsky Ed, W. F. *Introduction to Discrete Mathematics with Isetl.* Springer-Verlag New York Inc., (1996).
2. Ralph-J. Back, Joakim von Wright *Structured Derivations: a Method for Doing High School Mathematics Carefully. TUCS Technical Report Nr. 246, ISBN 962-12-0398-6* (1999).
3. Viera K. Proulx . The Role of Computer Science and Discrete Mathematics in the High School Curriculum *DIMACS Series in Discrete Mathematicsand Theoretical Computer Science, Volumen 36* (1997).
4. Ed Dubinsky, Philip Lewin . Reflective Abstraction and Mathematics Education: The Genetic Decomposition of Induction and Compactness *The Journal of Mathematical Behaviour, 55-92* (1986).