Mathematics and MateFun, a natural way to introduce programming into school

1st Sylvia da Rosa
Instituto de Computacion
Facultad de Ingenieria - UDELAR
Montevideo, Uruguay
darosa@fing.edu.uy

2nd Marcos Viera
Instituto de Computacion
Facultad de Ingenieria - UDELAR
Montevideo, Uruguay
mviera@fing.edu.uy

3rd Juan García-Garland
Instituto de Computacion
Facultad de Ingenieria - UDELAR
Montevideo, Uruguay
jpgarcia@fing.edu.uy

***Abstract*—In this paper, we describe an activity carried out with mathematics teachers of high school. The general objectives are: on the one hand, to include a stage of programming in the process of solving problems, emphasising the importance of discrete mathematics and logic to the training of all students. On the other hand, opening possibilities of introducing basic knowledge of programming into schools, an issue with high impact in higher education in computing. Finally, to introduce a functional language -MateFun- as a tool for achieving the said goals. The activity is organised in two parts: in the first one teachers are taught to program solutions to problems taken from their courses in MateFun; in the second part, the teachers choose a subject and design didactic instances of teaching it to their students using MateFun. In this way, the learning of programming is naturally presented to students as part of the mathematics course, integrating theoretical and empirical foundations of the concepts.**

***Index Terms*—mathematics, programming, learning**

I. Introduction: the didactic problems

We have participated for more than ten years in teaching the first programming course of the Computer Engineering career at the Faculty of Engineering in Uruguay. According to our experience, it can be said that the origin of most of the difficulties in learning programming lies in the mathematical background of the students. In general, the students entering the University have serious difficulties to understand concepts related to logic and discrete mathematics. The pervasiveness of continuous mathematics makes students (and teachers!) develop preconceived ideas with unfortunate consequences, especially for computer science education. For instance, they (almost) never describe a function including domain and range (all is $\mathbb{R}$) and they are restricted in their understanding of function to the idea of an algebraic formula for computing values; they hardly ever use either quantifiers carefully in proofs or notions of logic in calculations (i.e. boolean functions); and they treat induction as a recipe to solve certain problems (often involving sums of numbers).

Researchers and teachers of computer science of several countries' Universities have made the problem of providing high school students (and even primary school students) with basic programming knowledge their own [1]–[5].

The difficulties are a consequence of didactic approaches stressing the development of students' calculation skills. However, mathematics courses of high school are sources of several types of algorithmic problems, a fact that can favour the introduction of programming learning, creating a context where mathematics becomes more useful and applicable for students [6]–[9].

In this paper we present a didactics focusing on students' understanding of the motivations and the reasons of the symbolic manipulation using *functional programming*. Several authors agree that this programming paradigm is a good vehicle towards to enhance mathematics learning [6], [10], [11]. We have developed a functional language -MateFun [12], [13]- with the specific purpose of being a tool to learn mathematics at high school.

Our didactics is described through an activity with mathematics teachers, consisting in two parts: in the first one, they are taught to program solutions to problems taken from their courses in MateFun. In the second part the teachers choose a topic, design didactic instances of teaching it and put these into practice with their students using MateFun.

The paper is organised as follows: the main features of MateFun are briefly described in Section II. The first part of the activity with mathematics teachers (the course) is described in Section III and the teachers activities with their students in Section IV. Finally, we conclude and present some future work.

II. MateFun

The language MateFun is a functional programming language developed in our institute by the research group of the authors Viera and García-Garland, with the specific purpose of being a tool to support mathematical learning, especially in high school. Both in its creation and in its evolution, the opinions and comments of mathematics teachers are taken into account.

A. Description

MateFun is purely functional, meaning that functions do not introduce side effects and they only depend on their arguments. To be easily approachable it is available as a web integrated development environment (Matefun IDE[1]), as shown in Figure 1. The left frame is a text editor, where the

[1]https://matefun.math.psico.edu.uy

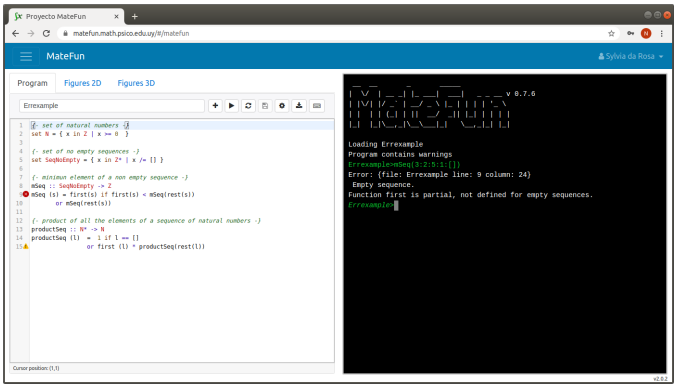

Figure 1. MateFun IDE

program is written, and the right frame is a shell with a read-eval-print-loop, where the expressions are evaluated.

Syntax and semantics of MateFun are both influenced by the seek to be a tool to express mathematics. The syntax is minimal and close to the usual mathematical notation. Semantically, it has the peculiarity of being strongly typed, while having no type inference. The skill to specify the domain and range of a function is part of the learning process when learning about functions. In MateFun type information must be given by users, and types in MateFun are called sets.

A MateFun script is a list of definitions of *sets* and *functions* over such sets. Predefined sets such as `R` (representing real numbers) or `Z` (representing integer numbers) are available as built-in constructs.

The user can define new sets either by comprehension or by extension just as usually presented in mathematics courses. In the following example we define the sets of natural numbers (`N`), non-zero real numbers (`Rno0`) and days of the week (`Day`):

```
set N    = { x in Z | x >= 0 }
set Rno0 = { x in R | x /= 0 }
set Day  = { Mon, Tue, Wed, Thu
           , Fri, Sat, Sun }
```

Sets such as `Rno0`, defined by comprehension, take a base set (`R` in this case) and refine it with a predicate. Predicates can be built by relational operators and from other predicates by conjunctions.

Functions are defined giving a signature and a proper definition. For instance, one could define the inverse function over the non-null real numbers:

```
inv :: Rno0 -> R
inv (x) = 1/x
```

MateFun supports some of the idioms used to define functions in mathematics. For instance, piece-wise functions can be defined, while, unlike most functional languages, it does not support pattern matching or conditional expressions. The following MateFun definition specifies the absolute value function over the real numbers:

```
abs :: R -> R
abs (x) =   x if x >= 0
        or -x
```

This program resembles the definition in the usual mathematical notation given by:

$$abs : \mathbb{R} \to \mathbb{R}$$
$$abs(x) = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{otherwise} \end{cases}$$

To emphasise that not all functions are numeric, MateFun allows to define non-numeric sets (eg. `Day`) and functions between those sets, such as `nextDay`:

```
nextDay :: Day -> Day
nextDay (d) =  Tue if d == Mon
            or Wed if d == Tue
            or Thu if d == Wed
            or Fri if d == Thu
            or Sat if d == Fri
            or Sun if d == Sat
            or Mon
```

Functions with multiple variables can be defined using *n-tuples* (the generalisation of Cartesian products). The following function computes the area of a triangle, given its base and height:

```
areaTria :: R X R -> R
areaTria (b, h)  = b * h / 2
```

We can also define *sequences* of elements of a given set; usually called *lists* in programming. The sequence set `A*` is defined inductively as:

- `[]`, the empty sequence
- `a:as`, a sequence composed by an element `a` belonging to `A` and a sequence `as` belonging to `A*`.

For instance, `N*` is the set of sequences of natural numbers.

There exist some primitive functions to operate with sequences: `first(s)` returns the first element of the sequence `s`, `rest(s)` returns the sequence `s` without its first element, and `range(n,m,k)` returns a sequence of numbers $(n, n+k, n+2k, ...)$ from `n` to `m` with step `k`. With `range`, combined with a function to `sum` the elements of a sequence, we can for instance easily implement the summatory $\sum_{i=m}^{n} i$.

```
summatory :: N X N -> N
summatory (m, n) = sum(range(m, n, 1))

sum :: R* -> R
sum (xs) =  0 if xs == []
         or first(xs) + sum(rest(xs))
```

Notice the use of recursion in the implementation of `sum`.

The language includes the primitive sets `Figure` and `Color`, and a set of primitive functions to create and transform *figures*. For example, the following function returns a red-coloured circle of a given radius, centred in the $(0, 0)$ point of a Cartesian plane.

```
redCirc :: R -> Fig
redCirc (r) = color(circ(r), Red)
```

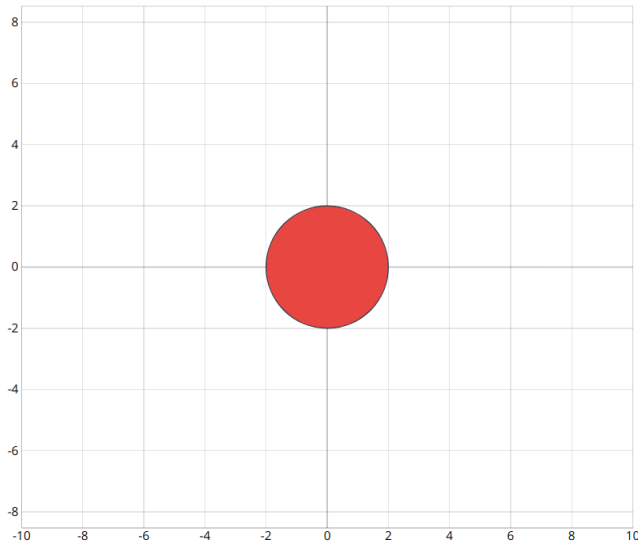

Figure 2. `redCirc(2)`

In MateFun, *animations* are sequences of figures. The following function takes a figure and a number `n` of steps, and returns an animation in which the figure is moved `n` times one unit to the right on the x-axis:

```
28 moveRight :: Fig X Z -> Fig*
29 moveRight (f, n)
30   =  [] if n == 0
31   or f : moveRight(move(f, (1, 0)), n - 1)
```

B. Evaluation

Matefun IDE provides a read-eval-print-loop interpreter where scripts can be loaded and evaluated. Then, if all the code of the previous subsection is contained in a file `Example` and we load it, the interpreter can evaluate expressions involving all the sets and functions defined in the file. This is represented in the prompt of the interpreter:

```
Example>
```

For instance, consider the function `abs`. In the shell we can ask to compute the absolute value of the number `10` by typing the expression:

```
Example>abs(-10)
10
```

Expressions may be function applications as in the previous case or any kind of combination of builtin operations, literals, and function applications. For example, we can evaluate:

```
Example>f(abs(-4)) + 5
5.25
```

We can also evaluate expressions that return figures, like:

```
Example>redCirc(2)
```

In this case the result is drawn in the "Figures" tab in the left frame of the IDE. Figure 2 shows the result of the example.

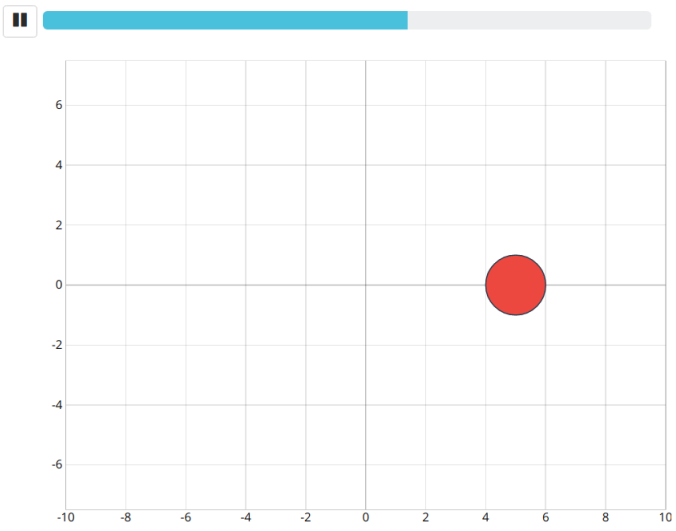

Figure 3. `moveRight(redCirc(1),10)`

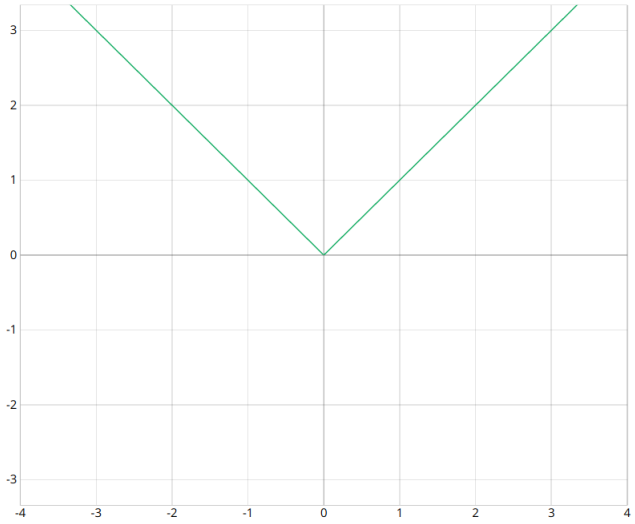

Figure 4. Function plot: `?plot abs`

In the case of animations, a timeline displays the progress of the frames. Figure 3 shows the sixth frame of the following animation:

```
Example>moveRight(redCirc(1),10)
```

The interpreter evaluates expressions and interprets special commands. For instance, we can graph a (one-variable) function using the command `?plot`.

```
Example>?plot abs
```

The result is shown in Figure 4.

C. Detecting Errors

Expressions that we try to evaluate in the interpreter may be ill-formed. For example, we could try to apply the function `abs` to an element of a different set than its domain. In this case, MateFun displays a message indicating the error.

```
Example>abs(Tue)
Error: {Interpreter column: 5}
Expected elements of R but found
Day.
```

We can also make mistakes by violating a constraint required by a set comprehension:

```
Example> inv(0)
Error: {Interpreter column: 2}
Value 0 does not belong to set Rno0
because the following condition is
false: [0 /= 0].
```

We can also make mistakes when defining functions. Suppose we change the signature of `abs` defined in line 8 of `Example` to:

```
8  abs :: R -> Day
```

In this case, two errors are found, in lines 9 and 10, when we try to return an element of `R`, while the function is defined to return a `Day`.

```
Error: {file: Example line: 9
column: 13}
Expected elements of Day but found
R.
Error: {file: Example line: 10
column: 12}
Expected elements of Day but found
R.
No File>
```

All previous errors are identified and reported as we try to load the file.

There are some cases where MateFun does not consider this kind of type mismatch as an error, but as a possible source of errors. An example is when we change the signature of `abs` defined in line 8 of `Example` to:

```
8  abs :: R -> Rno0
```

When loading the file, MateFun displays warnings instead of errors, and the process succeeds. It requires non trivial automatic reasoning to decide statically (i.e, when loading the module) if `abs` is well-typed. Our current approach is to delay errors and decide dynamically (i.e. in running time) in this cases. Also, this approach has the flexibility of allowing the use of functions whose codomain is included strictly in the potential range of the defining expression is useful. For instance, to avoid having to redefine the numerical operators (+, -, etc.) that are real valued functions, but also closed in sets such as `Z`.

```
Warning: {file: Example line: 9
column: 13}
Set Rno0 required is a subset of
resulting R. There is a chance that
its value is out of the set.
Warning: {file: Example line: 10
column: 12}
Set Rno0 required is a subset of
resulting R. There is a chance that
its value is out of the set.
Example>
```

Then, errors can occur when evaluating an expression. When these dynamic errors occur, MateFun messages point to the place in the code that generated them.

```
Example> abs(0)
Error: {file: Example line: 9
column: 13}
Value 0 does not belong to set Rno0
because the following condition is
false: [0 /= 0].
```

III. First part of the activity: teachers course

The activities were carried out using the moodle platform, during three months in which four face to face instances of four hours each took place. Teachers from diverse educational centres in different places of Uruguay participated. They teach in different high school years, and most of them teach to more than one group of students. After three face to face instances the teachers worked with their students during six weeks. The teachers were assisted through the moodle forums when working with their students. Some examples of that part of the activity are included in the next section. The fourth face to face instance was for the presentation of the final work.

The main objectives of the course for the teachers can be summarised as follows:

- Emphasise the relationship between mathematics and computer science, through reinforcing the importance of *discrete mathematics and logic* to understanding definitions, properties and proofs.
- Evidence the rigour attained through programming, and especially the benefits of the *functional paradigm and the language MateFun* for the learning of mathematics.
- Encourage *good programming practices*, through mathematically solving problems and then programming the solution, in a process that is not simply making programs work

It is worth noting that while this is the first time MateFun has been used, the course has been taught with other languages for several years (ISetL [14] 2006-2008, Python [2] 2013-2018). The accumulated experience has taught us that some of teachers' practices are part of the didactic problems raised in the introduction. One of these is the fact that mathematics teachers are not used to rigorously formulate problems;

[2]https://www.python.org/

many times they state problems forgetting details that they unconsciously interpret. The step of writing the solution in a rigorous language such as a programming language, and the step of executing the program, are in dialectic relation with the understanding of the problem. For example, if the problem is to find the multiples of a natural number, programming a function for the solution forces to explicitly including the natural number *and* a limit as input. In turn, this means that the problem must be formulated in terms of input-output [15], whether we are going to solve it or whether we are going to pose it to the students. In this sense, MateFun is a more powerful tool than the languages previously used, since it is strongly typed and it is necessary to write the signature of the function when programming it. However, whatever the programming language used, this fact has as negative side, as teachers comment (see Section IV): some details, irrelevant with paper and pencil work, require especial attention, like handling the computer, finding symbols on the keyboard, the syntax, etc.

A. *Examples of exercises*

Some examples of the teachers' work are selected to illustrate the approach towards the mentioned objectives. One of the themes presenting difficulties to students is recursive definitions of functions, as pointed out in teachers' comments in Section IV. At high school level it is clearly revealed when applied to define functions over natural numbers and sequences (the most adequate for beginners). The following are examples of basic exercises solved by the teachers.

```
{- non empty sequences of integers -}
set SeqNoEmpty = {x in Z* | x /= []}

{- the last element of a sequence of
integers-}
last :: SeqNoEmpty -> Z
last(l) =  first(l) if rest(l)==[]
        or last(rest(l))

{- minimun value of a sequence of integers
-}
minSeq :: SeqNoEmpty -> Z
minSeq(l) =  first(l) if rest(l)==[]
          or min(first(l), minSeq(rest(l)))

{- sum and product of the elements of a
sequence -}
sumSeq :: Z* -> Z
sumSeq (seq)
 =  0 if seq == []
 or first(seq) + sumSeq(rest(seq))

productSeq :: Z* -> Z
productSeq (l)
 =  1 if l == []
 or first (l) * productSeq(rest(l))
```

One of MateFun's features that teachers find more friendly, is the the way that errors messages are displayed. The message attempts to guide the user (teachers and students) for themselves discovering where the error is generated. Not only the error message appears in the section of the interpreter, but also the instruction of the program is marked with a symbol, depending on whether it is an error or a warning.

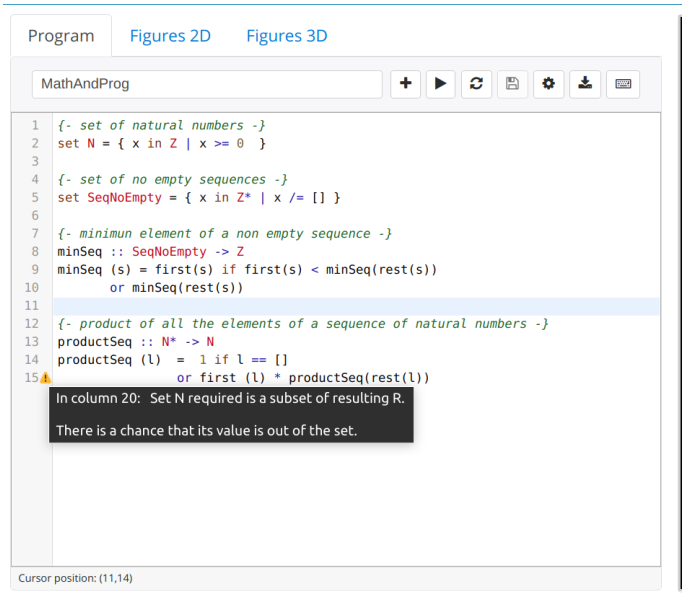

Figure 5. Example of a teacher's code with errors

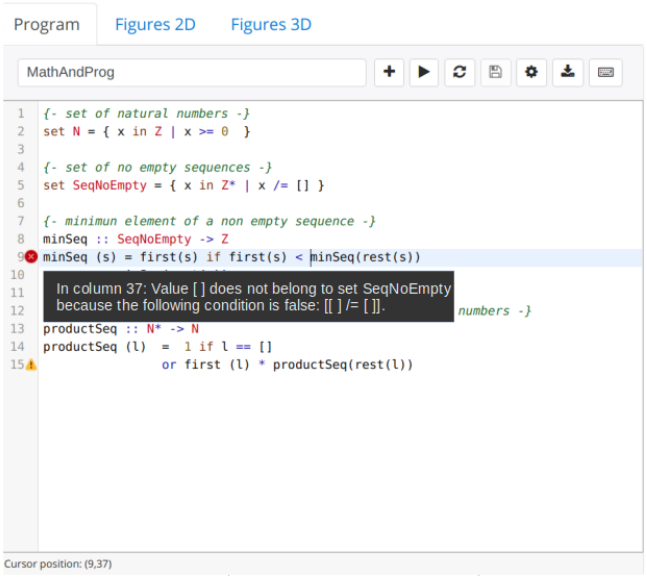

Figure 6. Example of a teacher's code with error exposed

In Figure 5 we can see the code written by a teacher to solve the basic exercises. We can see that there is a warning introduced by the use of the operator `*`, which returns elements of `R`. In fact this will not be a problem, because the operands will always belong to `N` in this function. A real mistake of this code appears if we, for example, try to evaluate `minSeq(1:2:[])`, because this call to the function will cause a call to `minSeq([])`:

```
MathAndProg> minSeq(1:2:[])
Error: {Interpreter file:
MathAndProg line: 9 column: 37}
Value [] does not belong to set
SeqNoEmpty because the following
condition is false: [[] /= []].
```

Figure 6 shows how the place in the code where this error occurs is pointed by MateFun's IDE.

In our course the teachers are asked to solve many problems involving programming of functions over sequences, using recursion and/or composition of functions. The example below shows a MateFun program for the factorial function (`fact`). This is a well known definition by the teachers and all of them succeed in solving this problem.

```
fact :: N -> N
fact (n) =  1 if n == 0
         or n * fact(n-1)
```

Then the teachers are asked to write another MateFun program (`factorial`) for the same function using two functions: `productSeq` defined above, and `range` introduced in Section II.

```
factorial :: N -> N
factorial (n) =  1 if n == 0
              or productSeq(range(1, n, 1))
```

Most of teachers arrived to a solution similar to that above, but with an extra redundant equation, which is used for a case that is actually encompassed for the definitions of functions `range` (case b < a) and `productSeq` (`productSeq([]) = 1`). This kind of errors in which edge cases are mishandled are frequently made by both teachers and students.

Several lessons are learnt from solving those kind of exercises. One of the most important is that a program perhaps gives the correct result (teachers' solution of factorial does), but has errors anyway. From the point of view of programming it is an error to make the computer do things that are not necessary or are redundant. Another one is that problems have to be precisely formulated: if the relation between `a` and `b` in function `range` is not specified as it is, the result could vary. Besides, programming more than one solution to a problem gives us the opportunity of introducing teachers to a topic that brings mathematics and functional programming even closer: the use of the principle of *structural induction* to prove properties of programs. Although the proofs are done with paper and pencil, it is possible to do them using equation reasoning (substituting equals for equals where every step has to be well founded) since MateFun is a pure functional language (without side effects).

In the next example, we present the proof of the following property:

Property. $\forall$ `n` $\in$ `N`, `fact(n) = factorial(n)`.

The property is proved using the principle of structural induction:

1) Base case: prove Property for `n` = 0
2) Inductive case: $\forall$ `n` $\geq$ 0, if `fact(n) = factorial(n)` then `fact (n+1) = factorial(n+1)`
3) If 1) and 2) then Property holds.

Base case:

$$\begin{aligned} & \texttt{fact}(0) \\ =\ & 1 && \{ \text{ def. } \texttt{fact} \} \\ =\ & \texttt{factorial}(0) && \{ \text{ def. } \texttt{factorial} \} \end{aligned}$$

Inductive case:

$$\begin{aligned} & \texttt{fact}(n+1) \\ =\ & (n+1) * \texttt{fact}(n+1-1) && \{ \text{ def. } \texttt{fact} \} \\ =\ & (n+1) * \texttt{fact}(n) && \{ \text{ arithmetic } \} \\ =\ & (n+1) * \texttt{factorial}(n) && \{ \text{ Ind. Hypothesis } \} \\ =\ & (n+1) * \texttt{prodSeq}(\texttt{range}(1,n,1)) && \{ \text{ def. } \texttt{factorial} \} \\ =\ & \texttt{prodSeq}(\texttt{range}(1,n+1,1)) && \{ \text{ lemma } \} \\ =\ & \texttt{factorial}(n+1) && \{ \text{ def. } \texttt{factorial} \} \end{aligned}$$

The lemma whose proof is left to the reader is: $\forall$ `n` $\in$ `N`, `(n+1) * prodSeq(range(1,n,1)) = prodSeq(range(1,n+1,1))`.

Since 1) and 2) hold then 3) holds, that is, the Property holds and the two definitions of the function are equivalent.

It is worth saying that traditionally, the principle of induction is used in a very restricted way in high school education, usually making no sense for students. Here the relevance of this method becomes evident.

This kind of exercises reinforces the importance of logic to understand definitions, properties and proofs which is one of the goals of the course as above said.

B. An advanced example

Interesting examples arose from concerns of some teachers and were discussed in the forum. In this section one of them is presented.

The teachers usually work with GeoGebra [16] that is an interactive geometry, algebra, statistics and calculus application, intended for learning and teaching mathematics and science from primary school to university level. They find a challenge in solving exercises that they have previously solved with GeoGebra. Some teachers give a great importance to the possibility of solving exercises and drawing the solution as they do with GeoGebra. We point out that a programming language like MateFun allows to construct the mathematical solution as an object - *the program* - putting into practice from more basic notions to advanced concepts. Even more, programs can be executed and we can see our solutions in action. The example below illustrates the case.

Given a function f it is possible to compute an approximation of the definite integral

$$\int_a^b f(x)dx$$

using the *rectangle method* (or *Riemann sum*). The method consists of partitioning the interval $[a, b]$ in n equidistant sub-intervals $[x_i, x_{i+1}]$ where $i \in \{0 \dots n\}$ $x_i = a + i\left(\frac{b-a}{n}\right)$. For each sub-interval a point x^* is chosen. A rectangle of width $\frac{b-a}{n}$ and height $f(x^*)$ is an approximation of $\int_{x_i}^{x_{i+1}} f(x)dx$ and the sum of rectangles approximates the full integral, improving with a bigger n.

Taking the leftmost point at each interval (left Riemann sum) the approximation is given by the expression:

$$\sum_{i=1}^{n} f(a + (i-1) * w) * w$$

where $w = \frac{b-a}{n}$

The function $f(x) = \frac{x^2}{10}$ is given. The problem is solved in MateFun in the following way.

```
f :: R -> R
f (x) = 0.1 * x ^ 2
```

The approximation of the integral is given by

```
integral_f :: R X R X N -> R
integral_f (a, b, n)
 = summatory_f(1, n, a, (b - a) / n)
```

where `summatory_f` is a function computing recursively the sum of the areas of the $n - i + 1$ rightmost rectangles, implemented as:

```
summatory_f :: N X N X R X R -> R
summatory_f (i, n, a, w)
  =  0 if i > n
  or f(a + (i - 1) * w) * w
     + summatory_f(i + 1, n, a, w)
```

Another version, with a more compositional style, using the function `sum` can be implemented. First, compute the sequence of areas:

```
areas :: R X R X R -> R*
areas (a,n,w)
 =  [] if n == 0
 or f(a) * w : areas(a + w, n - 1, w)
```

And then apply the known function `sum`.

```
integral :: R X R X N -> R
integral(a,b,n) = sum(areas(a, n, (b-a)/n))
```

It is possible to plot a representation of the rectangles used to approximate the area. The result for twenty rectangles in the interval 0-10 is showed in Figure 7.

This is accomplished by the following functions:

```

drawArea :: R X R X R -> Fig
drawArea (a, b, n)
 = joinFigs(rectangles(a, n, (b - a)/ n))

rectangles :: R X R X R -> Fig*
rectangles (a, n, w)
 =  [] if n == 0
 or rectBase(a, w, abs(f(a + w / 2)))
    : rectangles(a + w, n - 1, w)

```

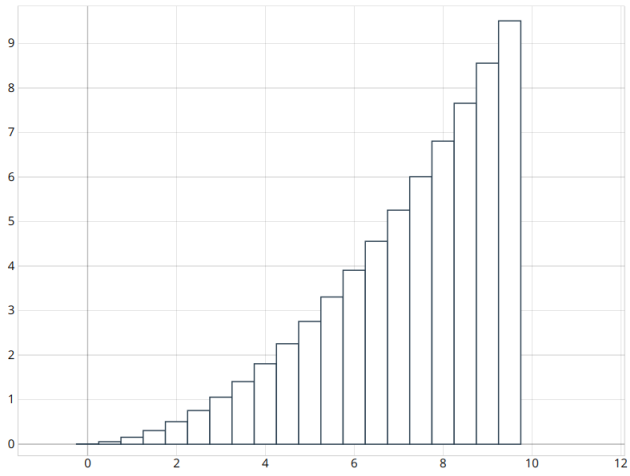

Figure 7. Integral: `(drawArea(0,10,20))`

```
rectBase :: R X R X R -> Fig
rectBase (i, b, h)
 = move(rect(b, h), (i, h / 2))
```

Through this example the power of MateFun can be appreciated with clarity: not only it makes possible to implement a solution of the problem using a syntax similar to mathematics (`integral_f` or `integral`), but also the visual representation of the solution can be programmed by the student. In other words, MateFun allows to program the graphic representations in contrast to other tools in which these are provided to the user.

IV. Second part: teachers activities with their students

In this section we present some exercises that teachers did with their students in the classroom. The themes that teachers choose vary according to the level of their groups (from 1st. to 6th. of high school). They have to take also into account the program of the school year. Popular themes are divisibility, lineal and quadratic equations, statistic, successions, analytic geometry, among others.

One of the topics that the teachers mentioned as successful is of successions. As an example we include an exercise done with fifty four students aged 15-16. The exercise is part of a task carried out over three sessions of 80 minutes each. It is formulated as follows:
Given the succession below:

$$a_n : \mathbb{N} \to \mathbb{R}$$

$$a_n = \begin{cases} 1 & \text{if } n = 1 \\ ((n-1)/n) * a_{n-1} & \text{otherwise} \end{cases}$$

1) write a MateFun function to obtain any term.
2) write a MateFun function that given a value `m`, returns a sequence with the first `m` terms of the succession.
3) write a MateFun function that given a value `m` calculates the sum of the first `m` terms of the succession.

Below are some of the solutions that teachers worked with the students in the classroom.

```
{-Part 1: any term of the succession -}
anyTerm :: N -> R
anyTerm (n) =  1 if n == 1
            or ((n-1) / n) * anyTerm(n-1)

{- Part 2: sequence of the first m terms
   of the succession -}
firstM :: N -> R*
firstM (m) =  anyTerm(m) : []  if m == 1
           or anyTerm(m) : firstM(m-1)

{- Part 3: sum of the first m terms
   of the succession.  -}
sumFirstM :: N -> R
sumFirstM (m) = sum(firstM(m))
```

This kind of exercises forces the students to write the definition of a succession as a function from $\mathbb{N}$ to any set; $\mathbb{R}$ in the case of the example.

The third part of the exercise was also solved with the formula below, which is traditionally presented to students.

$$\sum_{i=1}^{m} a_i = m * (a_1 + a_m)/2$$

Students were encouraged to implement it in MateFun and with teacher's help they arrived to:

```
sumFirstM' :: N -> R
sumFirstM' (m) = m * (anyTerm(1) +
                      anyTerm(m)) / 2
```

Since both definitions (`sumFirstM` and `sumFirstM'`) reveal two methods of calculating the sum of terms in a succession, in our course teachers were asked to prove that these are equivalent as it had been done for the functions `fact` and `factorial`. The following property was stated:

Property. $\forall$ `n` $\in \mathbb{N}$, `sumFirstM(n) = sumFirstM'(n)`.

The property was proved using the principle of structural induction:

1) Base case: prove property for `n = 1`
2) Inductive case: $\forall$ `n` > 1, if `sumFirstM(n) = sumFirstM'(n)` then `sumFirstM(n+1) = sumFirstM'(n+1)`
3) If 1) and 2) then Property holds.

Base case:

$$\begin{aligned}
&\texttt{sumFirstM}(1) \\
&= \texttt{sum}(\texttt{FirstM}(1)) && \{ \text{def. } \texttt{sumFirstM} \} \\
&= \texttt{sum}(\texttt{anyTerm}(1) : []) && \{ \text{def. } \texttt{FirstM} \} \\
&= \texttt{sum}(1 : []) && \{ \text{def. } \texttt{anyTerm} \} \\
&= \texttt{first}(1 : []) + \texttt{sum}(\texttt{rest}(1 : [])) && \{ \text{def. } \texttt{sum} \} \\
&= 1 + \texttt{sum}([]) && \{ \text{def. } (\texttt{first}, \texttt{rest}) \} \\
&= 1 + 0 && \{ \text{def. } \texttt{sum} \} \\
&= 1 * (1 + 1)/2 && \{ \text{arith} \} \\
&= 1 * (\texttt{anyTerm}(1) + \texttt{anyTerm}(1))/2 && \{ \text{def. } \texttt{anyTerm} \} \\
&= \texttt{sumFirstM}'(1) && \{ \text{def. } \texttt{anyTerm}' \}
\end{aligned}$$

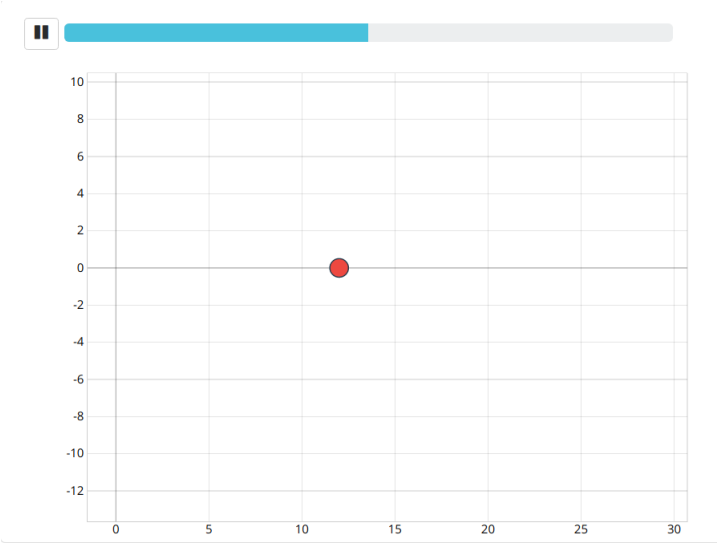

Figure 8. Animation: `move3cir(10)`

For space reasons the proof of the inductive case is not included. It remains as an exercise for the reader.

Through these exercises the teachers can experience one of the goals of the course with their own students, namely how the relationship between mathematics and computer science comes out in an easily understandable way. Even more, teachers could introduce their students into inductive proofs in cases they consider adequate.

In several researches on programming didactics we have argued that abstracting from concrete cases to obtain a generic one is not a trivial issue [7], [17]–[19]. As several authors also indicate [20]–[23], learning to program plays a fundamental role in training of abstraction from early stages. The following example shows the work that teachers did with their students in the classroom and the work we did with teachers in our course, related to abstraction.

A theme introduced by the teachers in a group of third-year high school students (aged 13-14) is about the multiples of a natural number. Teachers asked the students to program a function that makes a circle move *n* steps through the points of multiples of three on the x axis. A solution is presented below, in which the students adapted the function `moveRight` (see II) to program a function `move3` that moves a figure *n* steps through points corresponding to multiples of three on the x axis. They also programmed `move3cir` in which the parameter `fig` of `move3` is instantiated to a red circle previously defined. Notice that adapting `moveRight` induced the students to write `move3` abstracting the figure in the parameter `fig` as in `moveRight`.

```
circle :: R X Color -> Fig
circle (r, c) = color(circ(r), c)

move3 :: Fig X Z -> Fig*
move3(fig,n)
  =  [] if n < 0
  or fig : move3(move(fig, (3, 0)), n-1)
```

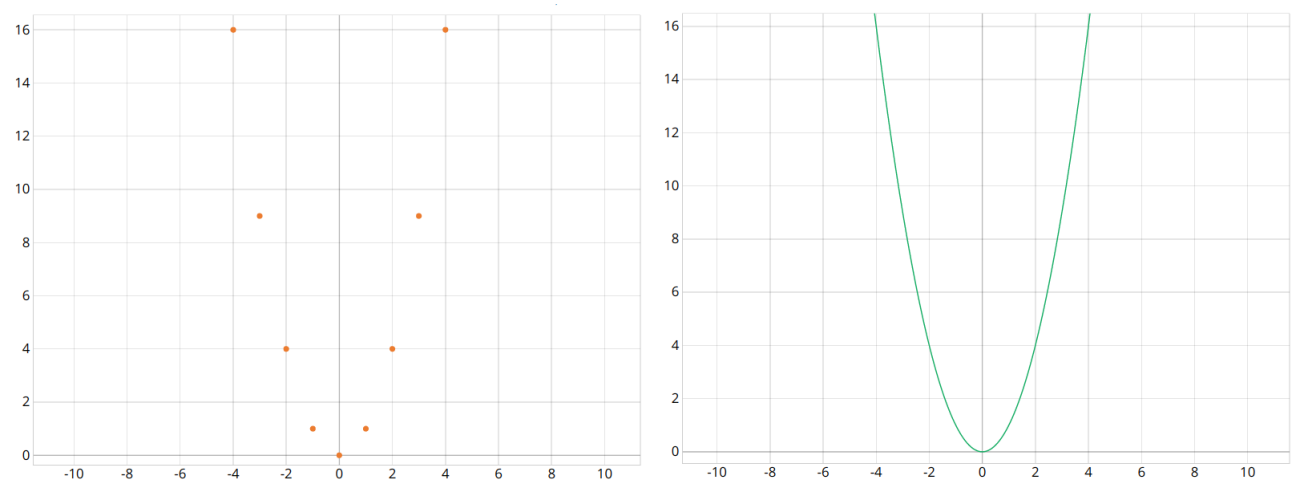

Figure 9. Parables in $\mathbb{Z}$ and $\mathbb{R}$

```
8
9  move3cir :: N -> Fig*
10 move3cir (n) = move3(circle(0.5, Red), n-1)
```

Since the problem asks for the multiples of three, they used the concrete case of the point `(3,0)` in the function `move3`. When the teachers presented this solution in our course they were asked how to generalise it to the multiples of *any natural number*, that is, abstracting the point (3,0) to `(num,0)`. To construct a general solution they observed that `move(fig,(num, 0))` moves the figure `fig` through the multiples of any natural number -represented by `num`- on x axis. The point discussed was how to define a single function that also performs the movement *n* times. Finally, teachers introduced the definition of `moveMultiples` below.

```
11 moveMultiples :: Fig X Z X N -> Fig*
12 moveMultiples(fig, n, num)
13   =  [] if n == 0
14   or fig : moveMultiples(move(fig,(num, 0))
15                          , n-1, num)
```

For instance, `moveMultiples(circle(0.5, Red), 5, 3)` moves the red circle through the multiples of three on the x axis, five times.

Teachers also noted the benefits of including the signature of functions as part of their definitions. This feature of the language can help students to surmount the restricted idea of a function as an algebraic formula for computing values, that we point out in Section I. Teachers presented the example below that shows two quadratic functions with the same algebraic formula defined over different numerical sets. Through their graphic representations (see Figure 9), the role of the signatures becomes evident.

```
16 parable1 :: Z -> Z
17 parable1(x) = x^2
18
19 parable2 :: R -> R
20 parable2(x) = x^2
```

In this exercise the students were asked to predict the result of calculations of several values for each case and then verify their responses with MateFun.

A. *Some students' work and teachers comments*

During the activities with their students teachers have observed interesting facts, some of which are summarised here.

The motivation of the students did not always coincide with the academic results: there were cases of highly motivated students who did not perform well and vice versa. However, compared to other groups where the proposal was not applied, a better understanding of the issues, greater concentration and predisposition to problem solving were observed.

The development of introductory activities to MateFun and its interface did not present major difficulties. The only thing worth noting is keyboard handling, working with parentheses, and symbols or characters that students could not find.

The problems started with the work of recursive functions. It was necessary to make explanations with simple cases such as defining the function for calculating the power of natural exponent, as shown below:

Question: Write a program in MateFun for the power function for natural numbers, according to our definition in which the base is greater than zero.

Student answer:

```
21 set NnoZero = { x in N | x > 0 }
22 pow :: NnoZero X N -> N
23 pow (x,y) =  1 if y == 0
24           or x * pow(x,y-1)
```

The work with graphs and figures was very attractive for the students and they were able to solve the activities without problems. An example of these is:

Question: Anticipate the result of this expression. Then, copy it in the MateFun interpreter and verify your answer.

```
> join( scale(circ(5),1/5)
      , move(circ(1), (0,3)))
```

Student answer: *First it will draw the circle to scale reducing its size to 1/5 of the original. Then it will draw a circle of the given radius centred on the point (0,3). Both will be drawn at the end of the operation.*

Many teachers emphasised the benefits of experience the dialectic relationship between *"the work with the pencil and programming in MateFun"*. This is an example of *a bridge between discourse and action* that Dowek mentions in [3] as one of the main contributions of programming to education.

V. Conclusions and further work

Although Computer Science (hereinafter CS) should be included in high school as an academic discipline taught by CS teachers, the difficulties still presented to do that [1]–[5], make this approximation adequate to provide students with basic programming knowledge. Even more, some of the cited authors suggest that teachers of other disciplines should be called for the task of introducing CS into school, especially the community of mathematics teachers [1], [4].

From a didactic point of view MateFun has shown to be a helpful tool for understanding the process of solving problems, without adding technical complications, compared with other tools. These are often software packages or apps not aimed to students' programming, like Geogebra, or programming languages of general purpose, like Python or Scratch[3]. Features like MateFun's type system including the signature of functions in their definitions; the possibility of programming visual representations; the error messages that mathematics teachers and students can easily understand, open possibilities to include programming into school. At the same time, most of teachers comments are encouraging with respect to the participation and motivation of students in solving the problems and exercises. However, evaluation instances of students performance is a further work issue.

Other lines of further work are mainly oriented on the one hand to the development of MateFun. In this sense, the activities with teachers and students are an excellent help. On the other hand, it is necessary to compile the activities carried out to obtain sufficient material and experience. The goal is to offer an integrated subject of mathematics and programming to teachers training, as a contribution for solving the didactic problems described in Section I.

[3]https://scratch.mit.edu/

References

[1] S. Peyton Jones, "Bringing Computer Science Back into Schools: Lessons from the UK," *SIGCSE'13*, 2013.

[2] P. Bradshaw and J. Woollard, "Computing at School: An Emergent Community of Practice for a Re-Emergent Subject," *In: International Conference on ICT in Education*, 2012.

[3] G. Dowek, "Quelle informatique enseigner au lycée?" *Bulletin de l'APMEP, nr. 480*, 2005.

[4] ——, "L'enseignement de l'informatique en France, Il est urgent de ne plus attendre," https://www.academie-sciences.fr/pdf/rapport/rads_0513.pdf, 2013, rapport de l'Académie des Sciences.

[5] "CSTA K12 Computer Science Standards," http://www.csteachers.org/?page=CSTA_Standards, 2011.

[6] C. Hall and J. O'Donnell, *Discrete Mathematics Using a Computer*. Springer, 2000.

[7] S. da Rosa, "Designing Algorithms in High School Mathematics," *Lecture Notes in Computer Science, vol. 3294, Springer-Verlag*, 2004.

[8] C. Lewis and N. Shah, "Building Upon and Enriching Grade Four Mathematics Standards with Programming Curriculum," *SIGCSE ACM*, 2012.

[9] P. Jansson, S. H. Einarsdóttir, and C. Ionescu, "Examples and Results from a BSc-level Course on Domain Specific Languages of Mathematics," *P. Achten, H. Miller (Eds.):Trends in Functional Programming in Education*, p. 79–90, 2019.

[10] C. d'Alves, T. Bouman, C. W. Schankula, J. Hogg, L. Noronha, E. Horsman, R. Siddiqui, and C. K. Anand, "Using Elm to Introduce Algebraic Thinking to K-8 Students," *S. Thompson (Ed.):Trends in Functional Programming in Education*, p. 18–36, 2018.

[11] P. Jansson and C. Ionescu, "Domain-Specific Languages of Mathematics: Presenting Mathematical Analysis Using Functional Programming," *J. Jeuring and J. McCarthy (Eds.): Trends in Functional Programming in Education*, p. 1–15, 2016.

[12] A. Carboni, V. Koleszar, G. Tejera, M. Viera, and J. Wagner, "Matefun: Functional programming and math with adolescents," in *Conferencia Latinoamericana de Informática (CLEI 2018) - SIESC*, 2018.

[13] G. Cameto, A. Carboni, V. Koleszar, M. Méndez, G. Tejera, M. Viera, and J. Wagner, "Using functional programming to promote math learning," in *2019 XIV Latin American Conference on Learning Technologies (LACLO)*, 2019, pp. 306–313.

[14] E. Dubinsky, "ISETL: A Programming Language for Learning Mathematics," http://www.math.bas.bg/softeng/bantchev/place/setl/isetl-for-mathematics.pdf, 1995.

[15] D. Harel and Y. Feldman, *AlgorithmicsThe Spirit of Computing*. Addison-Wesley. An imprint of Pearson Education Limited, 2004.

[16] M. Hohenwarter and K. Jones, "Ways of linking geometry and algebra, the case of geogebra," *Proceedings of the British Society for Research into Learning Mathematics*, vol. 27, no. 3, pp. 126–131, November 2007.

[17] S. da Rosa and A. Aguirre, "Students teach a computer how to play a game," *LNCS 11169: 11th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives, ISSEP 2018*, pp. 55–67, 2018.

[18] S. da Rosa, " The Learning of Recursive Algorithms from a Psychogenetic Perspective," *Proceedings of the 19th Annual Psychology of Programming Interest Group Workshop, Joensuu, Finland*, 2007.

[19] ——, "The construction of knowledge of basic algorithms and data structures by novice learners," *Proceedings of the 26th Annual Psychology of Programming Interest Group Workshop, Bournemouth, UK*, 2015.

[20] A. P. Ambrosio, L. da Silva, J. Macedo, and A. Franco, "Exploring Core Cognitive Skills of Computational Thinking," *Proceedings of the 25th Annual Psychology of Programming Interest Group Workshop, University of Sussex*, 2014.

[21] S. Papert, *Mindstorms - children, computers and powerful ideas*. Basic Books, Inc., Publishers / New York, 1980.

[22] A. V. Aho, "Computation and Computational Thinking," *The Computer Journal*, vol. 55, 2012.

[23] J. Wing, "Computational thinking and thinking about computing," *Philosophical transitions of the Royal Society*, vol. Phil. Trans. R. Soc. A 366, pp. 3717–3725, 2008.