Functions in computer science and in mathematics – ideas to bridge the gap

Reinhard Oldenburg
Augsburg University, Germany
reinhard.oldenburg@math.uni-augsburg.de

Abstract
Functions are a key concept both of mathematics and of computer science. As many students don't attend courses in computer science it is desirable to have learning opportunities at hand that give students a glimpse of an idea why this concept is of relevance beyond the narrow scope of classical high school math. We introduce several such contexts in which the power of functions in computer science becomes visible without putting too high demands on the students.

Introduction

The subjects of mathematics and computer science have a very different status. In Germany, e.g. math is compulsory for all students in all grades, computer science is just an add-on in many German states and it is not even offered in most grades. But even worse, even where it is offered, usually there are almost no links to mathematics lessons. This may be seen as the result of an emancipation of computer science from mathematics. Back in the 80s many computer science courses taught how to program elementary mathematical algorithms but finally computer science educators realized that this can hardly be the legitimation of a school subject and they defined computer science education independent of mathematical applications. Thus, nowadays there are almost no connections between the to subjects in German schools. I think that this situation should be changed so that students acquire a connected view of the world and important theories. Functions seem to be a topic that is so deeply rooted in both subjects that it can be used to serve as a bridge.

The function concept in computer science (e.g. Scott 2009) is, however, not a unique subject. Different programming environments conceptualize it in different fashions. Nevertheless, some common properties can be summarized as follows to bring out the similarities and differences to functions in mathematics. Similarities:

- Functions map some input to a unique result (although there are multi-valued functions in some languages).
- The result depends on the input (and in strict functional languages only on the input).
- Function can be seen on a higher level as objects to operate on to produce new functions, e.g. compose them.

Differences:

- Functions defined on numbers and have numbers as values are possible but many more functions are defined on other sets.
- Functions are computable functions in computer science.
- The sets involved are either finite or countable but finite due to computer memory limitations.
- There is no facility to reason about functions, eg. there is no automatic boolean valued function runs_in_finite_time(f).
- For many functions (at least in non-strict functional languages) the result depends not only on the input value. Some functions (such as that that gives the current time) don't have inputs at all.

Programming makes it easy to evaluate a function not only on a singe input but to apply it to all elements of a large (but finite!) set or list. This may seem like a trivial point but it is at the core of the question why computers are powerful.

A superficial characterization may be that in math functions are of theoretical interest, in computer science of practical interest. To broaden this view which is in fact too narrow, we show first theoretical and formal aspects of functions in computer science and later on demonstrate the practical relevance of functions in math - when they are applied. And for this application computer science is a fundamental tool.
Functional thinking concerns all thinking with regards to functions, variation and covariation of quantities and dependencies (e.g. Smith (2008)). In school it is mostly realized by the investigation of patterns (finding sequence of values) and by covariational thinking (e.g., "as x increases by one, y increases by three" (from Smith 2009)). Thus math courses mainly deal with the subset of function from numbers to numbers (which are represented by expressions, takes and graphs). Yet, there should be more to functions than this. There could be more, as we shall see.

Sound synthesis
In our first attempt to extend functional thinking we introduce functions of time that describe the variation of air pressure – more commonly known as sound. Sound waves passing a certain position in space (say one man's ear) change the pressure in a somewhat periodic way. A pure tone of frequency 440Hz is described by the function $0.5 \cdot sin(2\pi \cdot 440 \cdot t)$ where the 0.5 gives the amplitude (e.g. the volume). The Java applet http://myweb.rz.uni-augsburg.de/~oldenbre/webAU/index.html allows to enter such expressions and to hear what they describe. The range of the time variable $t$is from 0 to some specified maximum, e.g. 2 (seconds), i.e. $t \in [0,2]$. Then $0.5 \cdot t \cdot sin(2\pi \cdot 440 \cdot t)$is a sound that gets louder over time and $0.5 \cdot sin(2\pi \cdot (440 + 200 \cdot t) \cdot t)$is one that gets higher during the 2-second interval. What sound is encoded by $sin\big(2\pi \cdot \big(500 + 200sin(3 * t)\big) \cdot t\big)$?
This basic example shows two typical operations: synthesis (given the idea of a how a sound may be structured (such as "becoming louder") finding a functional expression) and analysis (given a function, imagine how the sound defined by it is). Modelling and realizing (or de-modeling) are this interwoven.

Digital Image processing
Digital images are encoded as a matrix of pixels. For grey scale images each pixel is characterized by a single number, its brightness (see e.g. (Burger&Burge 2011), (Oldenburg 2006)). Thus, a grey scale digital image is exactly the same as a matrix in mathematics. One may apply a function to each entry to increase or decrease brightness, contrast or even to turn an image to its negative. This involves only easy algebra. However, usually one has to write programs to do this. To eliminate this difficulty I wrote some Java applets that are accessible on my web page http://myweb.rz.uni-augsburg.de/~oldenbre/webBV/index.html to overcome this problem. Students can perform various operations by specifying the mathematics transformation functions. Examples are shown below in Fig. 1 and 2.

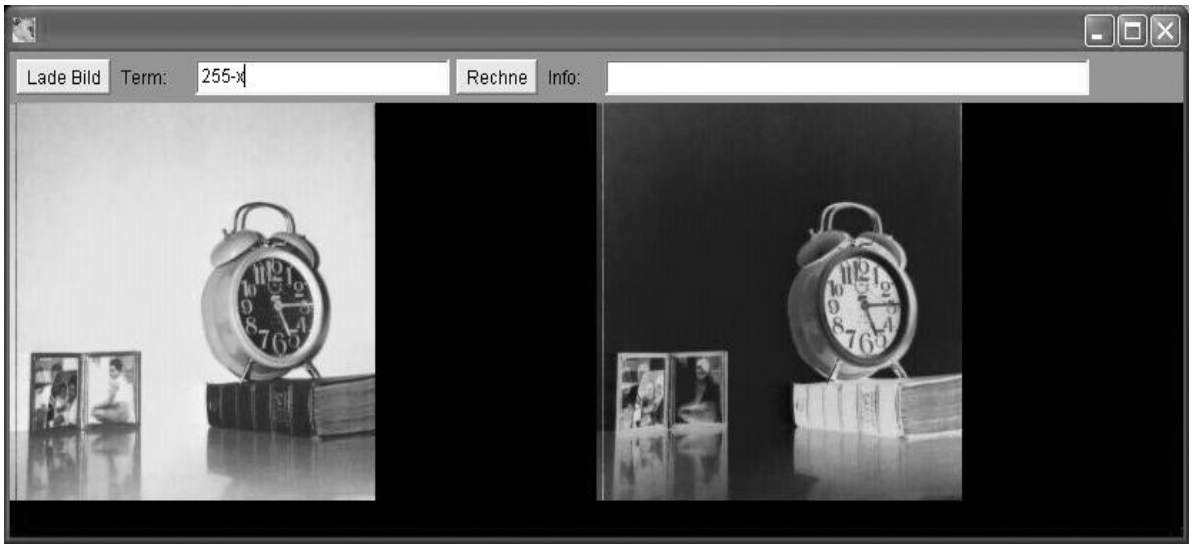

Fig 1: An Applet to transform the brightness of pixels according to a function

Fig 2: Example of a calculated local displacement

There are three types of operations accessible:

- Point operations: The only transformation in this case is to change the colour information of pixels. For grey scale images with brightness values from 0 (black) to 255 (white) one just defines a map that gives the new brightness in terms of the old one. An expression like $x + 50$makes the image brighter while $x/3 + 100$reduces the contrast. The functions defined by these expressions are applied simultaneously to all pixels in the image. For color images one hast to define functions of three variables (r, g, b) (red, green,blue) that gives the new colour information. The function $(r, g + 20, b)$slightly enhances the green component. By mixing colours various effects can be achieved.
- Geometrical transformations: One may leave the colour information as it is but move the pixels by specifying transformation. A pixel with coordinates (x, y)may move to $\big(x, y + 30 * sin(x/50)\big)$resulting in a waved image (Fig. 2). Of course, more simple things like translations, dilations and rotations can be realized as well.

- Image combination: Starting from two images (of the same size to make life a bit simpler) one may produce new ones e.g. by simply adding colour values. This results in composition of both pictures.

As with sound the functional view on images allows to synthesize and to analyse functions. However, it need some getting used to. A teacher said that he hardly recognized functions in this. Students, however, are much more direct, simply using the tool and thereby doing math.

Digital Video processing

Yet another transformation of functional thinking can be achieved by taking time as the domain of the function and images its range as it is possible with the applet http://myweb.rz.uni-augsburg.de/~oldenbre/webBV/index.html that takes the images from the computer's webcam (and hence requires the rights to do so, so that trying out may need some fiddling with different browsers and their settings). Functions of time are very commonplace, e.g. the temperature *T* varies in time *t* and this is easily written as a function *T*(*t*). Unusual is the range: We associate an image $B(t)$ to every moment in time. The variable t always stands for the current time (in seconds since the start of the applet). $B(t-1)$Gives the image taken at „now minus 1 second", i.e. 1 second back. $B(t+1)$is not implemented due to limited abilities to predict the future. $B(t/2)$Gives pictures at half of the original speed, a slow down effect. Students can analyse what a given function expression will do to the video stream and the other way round they can synthesize a function that realizes some effect.

Here are some interesting functions to analyse: $B(10-t)$, $B\big(5+5sin(t)\big)$. For what values of a do some parts of the film produced by $B\big(t+asin(t)\big)$go backwards in time? How much faster than the original are the fastest parts?

As described above, pictures may be pixel wise added or subtracted. This makes$B(t-1)+B(t)$or $\big(B(t-1)+B(t)\big)/2$ to show nice effects. Maybe even more summands may look good. The difference is of special interest: $B(t)-B(t-1)$or the difference quotient$\big(B(t)-B(t-0.1)\big)/0.1$. This shows where changes in the image occurred. The derivative thus is kind of change detector: It differs from 0 when we have great changes. Using such methods astronomers are looking for super nova explosions. More down to earth one may realize an anti-theft system based on this.

It is worth comparing $B(t/2)$ in this applet with the function graph if$f(x/2)$. How does the similarity express itself?

A mathematics lesson based on this may open the eyes that with a bit more of math and with fast computers one may automatically manipulate „live" images from some event, e.g. one may automatically remove unwanted statements from banners of the participants of a demonstration, as an example.

The examples up to now may run under the header of "the role of mathematics in the information and communication society". I think that it is an important goal to make students aware of the importance of math in all the digital tools that are omni-present in our society. The applets above address only a very small range of this.

Now we leave this area and complement it with a very different aspect of functions.

Formal aspects

This section makes two strong assumptions: That students have some proficiency in writing and reading programs in the Python language (which is, however, well suited for educational purposes, much more than e.g. Java) and that they are interested in a theoretical question: We show in a sketchy fashion how the theoretical construct of pure functions is sufficient to do all possible computations and moreover to create all objects that are computable at all!

In the Python programming language functions can be defines with the lambda key word as the following example shows:

```
>>> Plus1=lambda x: x+1
>>> Plus1(7)
8
```

This concept of functions comes in handy at places where a function is to be passed to another functions as in the following example to calculate an integral:

```
>>> integral(lambda x: x*x, 0,1)
```

The somewhat strange name lambda is due to a theory by Chruch, the so called lambda calculus (Michaelson 2011). In this calculus, lambda is the only things that exists from the beginning. All other things like numbers, lists, loops, etc are all defined in terms of lambda. We shall give an idea of how this works.

First, we aim at reducing the if-then-else construct of programming languages to basic logical conjunctions. These conjunctions, `and` and `or`, are first taken from Python. Later on we will show how they can be implemented.

The `if` function to be defined will be in a purely functional fashion, i.e. we'll have an expression of the form if *condition* then *expr1* else *expr2*. This will be based on logical operators as realized in Python. They have the nice property that everything that is not false is interpreted as true. Furthermore, *a* and *b* gives *a* if *a* is False and *b* if *a* is true (because in the case of a being true, the whole statements value is the same as that of *b*). Similarly, *a* or *b* gives *a* if *a* is not False and *b* otherwise. With this, we can replace if *condition* then *expr1* else *expr2* by (*condition* and *expr1)* or *expr2.*

An example (% is the modulo operator, i.e. `12%5` gives 2):

```
>>> isEven=lambda x: (x%2==0 and 'yes') or 'no'
>>> isEven(8)
'yes'
>>> isEven(9)
'no'
```

The factorial of a number can now be defined:

```
>>> fact=lambda n: (n==1 and 1) or n*fact(n-1)
>>> fact(6)
720
```

All loops can be reduced to recursion. This may not be convenient, but it is possible.

Up to now we have used boolean constants, numbers and operations from Python. Next we show that these can be defined in terms of lambda as well. True is simply a function of two alternatives which chooses the first one.

```
myTrue= lambda x,y: x
myFalse= lambda x,y: y
myAnd=lambda x,y:x(y,x)
myOr=lambda x,y:x(x,y)
myNot=lambda x:x(myFalse,myTrue)
myIf=lambda p,x,y:p(x,y)
```

So myTrue is the function that give its first argument, and myFalse gives its second and the working of myAnd and myOr are the same as the behaviour of its Python equivalent explained above. Lets define a conversion function to bring things into human readable form:

```
def asString(a): # convert back to Python objects
    if a==myTrue: return "TRUE"
    if a==myFalse: return "FALSE"
```

Now we can test deMorgans law:

```
for a in [myFalse,myTrue]:
    for b in [myFalse,myTrue]:
        L=myNot(myOr(a,b))
        R=myAnd(myNot(a),myNot(b))
        print("a="+asString(a)+" b="+asString(b)+
            " not(a or b)="+asString(L)+
            " not(a) and not(b)="+asString(R))
```

The output of this program is

```
'a= FALSE  b= FALSE  not(a or b)= TRUE  not(a) and not(b)= TRUE'
'a= FALSE  b= TRUE  not(a or b)= FALSE  not(a) and not(b)= FALSE'
'a= TRUE  b= FALSE  not(a or b)= FALSE  not(a) and not(b)= FALSE'
'a= TRUE  b= TRUE  not(a or b)= FALSE  not(a) and not(b)= FALSE'
```

The definition of numbers is done recursively along the lines of the Peano axioms by starting with the function that gives the identity function as zero and the successor of a function to be a function that iterates a given function one times more than its predecessor:

```
myZero=lambda f: lambda x: x
mySucc=lambda num: lambda f: lambda x: f(num(f)(x))
myAdd=lambda a,b: a(mySucc)(b)
myMul=lambda a,b: lambda f: a(b(f))
myPot=lambda a,b: b(a)

myOne=mySucc(myZero)
myTwo=mySucc(myOne)
myThree=mySucc(myTwo)
```

To convert between numbers in this sense and Python numbers we need to further functions:

```
def P2L(n): # Converts a Python number into a lambda number
    if n==0: return myZero
    return mySucc(P2L(n-1))
def L2P(num): # Converts lambda number to Python
    return num(lambda x: x+1)(0)
```

Now one can add e.g. `myAdd(myOne,myTwo)`. The result of this is a function of course, that can, however, be converted to 3 by applying the function `L2P` to it.

This concludes this theoretical part. It should be clear by now that all object and operations can be reduced to a simple concept, the lambda expressions that are simple functions. This allows to do general proofs about computability and gives a logical foundation of the importance of functions.

Experience

So far we have some experience in implementing this approach in the classroom. We have taught the image processing unit to various group ranging from weak 8th graders to average and above average 10th graders. The experience showed that even weaker students can work successfully with the environment and that there is an enormous potential for motivation of students. We have also taught computer science lessons in which bitmaps were manipulated by the algorithms described above (as well as some others) but within programs that students developed rather than in the web bases playground applets presented above. My impression is that this led students to recognize the role of mathematics in such applications as Photoshop – but we did not evaluate this by any means.

Similar statements hold for the sound applet as well which has been testes with 10th graders. We have not, however, taught the last theoretical section.

Conclusion

This paper has presented some new views on functions that can be taken in high school mathematics. These topics are quite accessible to high schools students and they have fun doing these activities. A more detailed analysis of how this sharpens the concept image of functions has to be carried out however.

Literature

Burger, W., Burge, M. (2011). Principles of Digital Image Processing: Fundamental Techniques. Heidelberg: Springer.

Michaelson, G. (2011). Functional programming through lambda calculus. New York: Dover.

Oldenburg, R. (2006). Die Mathematik der Bildverarbeitung. In: Istron Bd. 9. Hildesheim: Franzbecker.

Scott, M. L. (2009). Programming Language Pragmatics. Morgan Kaufmann.

Smith, E. (2008). Representational thinking as a framework for introducing functions in the elementary curriculum. In J. Kaput, D. Carraher, & M. Blanton (Eds.), Algebra in the EarlyGrades. Mahwah, NJ: Lawrence Erlbaum Associates/Taylor & Francis Group and National Council of Teachers of Mathematics.