

## Programming as a mathematical discipline

*We identify Mathematics with a human activity, with patterns of reasoning, with methods of exploiting our powers of abstraction, with traditions of mixing rigour with vagueness, with ways of finding solutions.*

- Edsger W. Dijkstra, EWD317

In this article, the nature of computations, programs and the programming task will be discussed. Also, the essence of a mathematical formalism is presented and the benefits of considering programs as formal objects worthy of mathematical treatment will be put forth.

The computations performed by an automatic computer are basically automated symbol manipulations. A computer at the most fundamental level basically manipulates one sequence of 1's and 0's to produce another sequence of 1's and 0's. And in order to perform meaningful manipulations, we write programs. A program can be thought of as a formula under the control of which a computer performs its manipulations. Or to look at it another way, using the words of Edsger Dijkstra, "A program is an abstract symbol manipulator, which is made concrete by supplying a computer to it".

Programming is a goal directed activity and a program alone is only half a conjecture. The other half of the conjecture is the specifications the program was designed to meet. A programmer can claim to have done his job only if he supplies the program, its specifications and a proof that the program meets its specifications.

Furthermore, the programs written by a programmer may be the last things to leave his hands, but the true subject matter of the programmer's trade are the computations evoked under the control of the programs he writes. He must have a good grip on the computations. But for most programs the number of computations performed are so large that it is impossible to imagine them executing in time, and therefore we must not try to imagine them.

In this connection, one must understand the futility of operational reasoning. By operational reasoning I mean the method of trying to justify or understand programs by imagining the steps they will take during the course of their execution. This way of thinking has two major drawbacks. Firstly, as mentioned earlier, the number of computations performed by most programs are beyond the scope of our imagination and hence we can imagine only a small subset of them at any given time. Trying to make assertions about our programs by taking only some subset of the possible computations into account allows us to make assertions about these computations only and not about all the possible computations. In my experience, the programmer usually leaves the remaining computa-

tions under the category of “they should work as well”. The amount spent on software maintenance – or “bug fixing” if you like – today confirms that this is generally not the case.

Secondly, and this is far more serious, operational reasoning places such a large demand on our inventive powers that it acts as a handicap when we try to develop new algorithms. I will come back to this later in the article.

In more general terms, when trying to make assertions about a large set of objects, it is far more effective to deal with the definition of the set rather than dealing with its individual members.

Turning our attention to formal mathematics, a calculus basically provides us with an appropriate level of abstraction such that we are able to focus on the essence of the problem directly through the notations of the calculus. The formal language and operators give us a concrete grip on concepts and the relationships among them. Furthermore, manipulations of the formulae of a calculus is a mechanical activity whose correctness is easily verifiable. All one has to do is check whether the manipulations satisfy the definitions of the operators of the calculus.

The un-interpreted manipulation of formulae is a great boon. It takes a huge load of our inventive powers viz. instead of thinking in terms of abstract concepts on which we have only a vague grip, we can now deal purely in terms of symbols and operators whose un-ambiguous definitions allow us to manipulate them in a rich variety of ways. It is worth mentioning that most of these manipulations lead to results that we could not have arrived at otherwise.

In his book, “A Discipline of Programming”, Edsger Dijkstra presents a formal system for deriving programs. It is more or less self-contained in that it builds on little more than the predicate calculus. The system provides a formal model of program execution and defines all programming language constructs formally. Using this system, we are able to make assertions about the definitions of the computations without ever having to consider individual computations.

One of the immediate benefits of this approach is that it presents a strictly non-operational approach to programming. The requirements of all algorithms can be represented in the formal language of the system. Once this is done, programming becomes very much like doing math viz. one can have a structure which one follows while solving the problem and within that structure one keeps trying various manipulations until the desired result is achieved. In the end, if one has obeyed the rules correctly, the resulting program is guaranteed to be correct. At no stage in this process is one required to think of the program executing in time.

Moreover, it has been observed that programs written with the aid of this system have been better than programs written without using it. A plausible explanation for this observation is that we are reducing the load placed on our inventive powers by increasing our calculational (read mechanical) load. Using this system, we get rid of the handicap of operational reasoning that I mentioned earlier.

Regarding programming as a mathematical discipline has tremendous benefits. The term “debugging” will become obsolete as our programs will be correct by construction. Finally, new algorithms are what allow us to apply computers to new domains. And using the appropriate mathematical tools, we should be able to invent really new algorithms. The prospects are exciting!

Mumbai, 5 October 2004

Apurva Mehta  
12B Woodlands  
67 Peddar Road  
Mumbai 400026  
India  
apurva@mathmeth.com