International Journal of Computing and Optimization
Vol. 4, 2017, no. 1, 187 - 192
HIKARI Ltd, www.m-hikari.com
https://doi.org/10.12988/ijco.2017.7913

A Mathematical Theorematic Approach to Computer Programming

Luis Valero-Elizondo

Math. Department
Universidad Michoacana
Edificio Alfa, C.U
Morelia, Michoacan, C.P. 58030, Mexico

Abstract

In this paper we explore similarities between computer programming and the creation and proof of mathematical theorems. We point out some practical advantages to this approach in creating large projects, so that they become easy to create, maintain and extend, and can also transport between different programming languages.

Keywords: computer programming, theorems, language design, modularity

1. Introduction: defining 'computer program'

When computers were firsts introduced, there was no such thing as software: a computer was made to carry out a specific task, and that was all it would do. This was extremely inconvenient. People quickly realized that it was much easier to create a generic computer capable of carrying out different basic tasks (commands) and then write “software” that joined these commands in order to simulate other computers: this is how programming came into existence. You can find a list of references for the History of Programming Languages in the Bibliography.

How is computer programming like mathematical theorems? In order to answer this question, we have to define 'computer program'. Here is one simple definition:

A computer program is a sequence of instructions that performs a certain task.
From a mathematician's perspective, this definition needs clarification: what is an 'instruction', and what is a 'task'? One can define 'instruction' as one of the many commands which are found in a given programming language, but here we would be forced to adopt a specific language and ignore all the others. This would probably alienate a large number of programmers of the languages we left out, so we must find a more general definition of 'instruction'. Here's a possible broad definition:

An instruction is one of the following:

1) another program (that is, a subroutine), which may be the original program (recursive call)

2) variable assignment

3) an if-then-else statement

4) a for-loop

5) a while-loop

6) a return statement

Here the if-then-else,, for-loop and while-loop all must use other programs in their definitions. We shall illustrate this in the Example section.

All major programming languages would have no problem with this definition of 'instruction'. We are only left with the definition of 'task', but it turns out this is not necessary: we could simply say that a computer program is a sequence of instructions (as defined above). Whether these instructions accomplish anything is irrelevant to the definition of computer program.

If we use these guidelines to write a useful program, we will inevitably reach a point where we must use a 'real' command, such as 'print' in python, or 'puts' in ruby, or 'echo' in php, or whatever it is you have to use in whatever language you use. As we mentioned before, we cannot choose a specific language - because that would destroy our universal portability - , so we must stop at this point and declare that the undefined programs are called 'Axioms'. The exact same thing happens in mathematics, where we reach statements that 'must be true but cannot be proved'.

2. Transporting the program to a programming language

In order to run this program, we must transport it to a real programming language, such as python, C, ruby, php, java, javascript, or whatever language you use, as long as it supports functions, variable assignment, return statements, if-then-else statements, and for- and while-loops. Here's what we must do:

1) Write each Axiom as a program in our specific language.

2) Explain how your language handles the instructions, that is, how to call a subroutine, assign variables, return values and perform if-then-else statements, for- and while-loops. For example, in php you use {and} in the if-then-else statements, but in python you use: and indentation.

3) Provide any constants that your programs need.

With this information, it is now possible to write a valid program in the language of your choice. And as you can see, with very little effort it is possible to use the same program in any other language.

3. Example

Let's write the usual 'Hello world!' program to illustrate this approach.

We write programs starting with the topmost routines. We use conventions from other languages to write functions, variables and statements, but these conventions are irrelevant.

```
program = HelloWorld(){
    var language = UserLanguage();
    if(IsSpanish(language)){
        Write(CONSTANT(1));
    } else{
        Write(CONSTANT(2));
    }
}
```

The programs UserLanguage, IsSpanish and Write can be declared as Axioms.
The function CONSTANT gives access to the constants used by this (and all) programs.

In order to write this program in php, we provide the following information:

Language of choice: PHP (to provide the syntax for the variables and the statements).

Axioms: they can be defined in php as follows:

```
function UserLanguage($language){
   $lang = substr($_SERVER['HTTP_ACCEPT_LANGUAGE'], 0, 2);
   return $lang;
}

function IsSpanish($lang){
  if('es' == $lang){
     return true;
  } else{
     return false;
  }
}

function Write($str){
  echo $str;
}
```

We must also provide the value of the constants:

```
CONSTANTS:
        1=> "¡Hola, mundo!',
       2 => 'Hello, world!'
```

But someone else could implement the same program in python, by specifying the corresponding information for python: the format for writing functions, variables, loops and conditionals, and also the definition of the Axioms using python. Or ruby, or java, or C, or Pascal, etc.

4. An application

This approach to writing computer programs has several advantages:

Your program is essentially illustrating the algorithm you are using, so you or anyone looking at your program will easily understand what it does in terms of other programs. This makes creating, maintaining and extending large projects much easier.

A large program can be divided into different programs, which can then be written by different teams. These teams can write their own code knowing exactly which programs (i.e. functions) they can share with other teams and which ones they are free to write for their own use only.

The people writing a program need not have a common programming language: you can have php and ruby programmers working on the same project, and once it is done, you can decide which language you want to use to implement it.

In fact, even people with no programming experience can 'write programs', since they basically only have to design the algorithm that shapes the program. In particular, it would be very easy to ask non-programmers to describe what they want done and use that as the starting point for the program: '*Write a program that will diagnose a patient's disease based on their symptons, their medical history and recent events.'* will become a program Diagnose with input variables patient_symptons, patient_history and recent_events, and these variables behave in ways that can also be described by the medical expert even if they are completely computer-illiterate. This would also minimize the time it takes to create a project, since the client would already provide the essential algorithm.

5. An actual website based on this approach

In an ideal world, there would be a database where programmers from all over the world and all programming languages would be able to search and contribute their own programs. However, we don't live in an ideal world, and many software-writing companies would find such a universal database undesirable. A more appealing option is to create a private database of programs, where your programmers - regardless of their language of expertise - can search for and create programs.
A database of this kind will also have advantages of its own (in addition to the ones listed before).

You can easily track the progress of your projects, by looking at the axioms of each project at any given moment.

You can exchange programs with other companies who use a compatible database.

You can search your collection of programs. For example, it would be easy to find all the programs that use a particular program that logs in a user.

You can implement tools to help write your programs, for example, by keeping track of the available variables at each point.

Creating such a website would be a matter of hours for an experienced programmer.

6. Conclusions

The way computer programming is typically done is like proving the fundamental theorem of calculus directly from the Zermelo-Fraenkel axioms of set theory.

Such a proof may be correct, but it would be extremely difficult to write, very hard to follow, and all but impossible to use to prove other theorems. Mathematicians have found better ways to prove theorems (after all, they have been writing proofs for thousands of years, since Euclid's time), so it's a good idea to use some of their techniques when writing programs. By writing programs in a theorematic way, we let the machine handle most of the work in writing a program, and can concentrate on what the program must do.

References

[1] Association for Computing Machinery (ACM) SIGPLAN, *A History of Programming Languages: Proceedings of the ACM Conference on Programming Languages*. Academic Press, Los Angeles, Calif, 1978.

[2] Paul E Ceruzzi, *A History of Modern Computing*, Cambridge, MIT Press, second edition 2003.

[3] Donald E. Knuth, Luis Trabb Pardo, Early development of programming languages, *Encyclopedia of Computer Science and Technology*, Vol. 7, Marcel Dekker, 419–493.

[4] Richard Wexelblat ed., *History of Programming Languages*, Academic Press, New York, 1981.

Received: October 12, 2017; Published: November 17, 2017