A categorial mistake in the formal verification debate

extended abstract

Javier Blanco Pio García
Universidad Nacional de Córdoba
Argentina

Abstract

The problem of problem verification has been discussed from a practical and a theoretical perspective. This debate seems very much alive (see [Ede07]). In the article [DMLP79] De Millo et al. have defined the agenda of the following debate. They claim that program verification is an impossible and an undesirable goal for computer science, because the community of computational scientist does not have an effective way to verify or accept theorems. Fetzer in [Fet88] has proposed an answer to this article by making a distinction between programs and algorithms and by identifying the specification stage as the target of testing programs. This distinction is grounded in the dichotomy between causal processes and logical structures. Algorithms can be verified because of their logical nature, but programs, as long as they can be executed in a physical machine, have a causal character. We suggest that the distinction between program and algorithm could not been made in Fetzer's terms. An ontological division between programs and algorithms condemn the possibility of making any verification at all. Instead we propose that we can speak of causal and logical aspects when we are dealing with programs. This ontological deflation has as a consequence a redefinition of the testing target: the modeling process.

The notion of formal verification of programs has been one of the main issues in the debate about the nature of computer or computing science. Whether one considers formal verification possible or not implies to take a position concerning the relationship between computing science and mathematics. This relationship may be studied from the point of view of what computing science is but also of what it should be. Two somewhat "extreme" positions in this debate are Hoare's who considers that "computer programs are mathematical expressions" ([Hoa86], p. 115), and De Millo et al. [DMLP79] who claim that program verification is an impossible and undesirable goal for computing science. This last article started a long debate which seems far from being settled. The main arguments they present against program verification are practical and social. Many of these arguments should be reconsidered in the light of the new achievements in program verification (and program derivation) tools, but we will not do it here. Among the answers to De Millo et al. the one by Fetzer is particularly interesting, since he attempts to reach a similar conclusion (the impossibility of program verification) but by a completely different approach. He claims that trying to verify programs is a categorial mistake, that programs as such cannot be verified since they are causal rather than mathematical structures. Fetzer distinguishes algorithms -which are indeed mathematical expressions and hence suitable to mathematical or logical verification- and programs which correctness can only be tackled by using empirical methods.

We claim that Fetzer has misidentified the problematic areas in program verification. According to Fetzer, there is a side in programs that has a causal nature, so it falls outside the scope of logical methods. On these grounds Fetzer builds a distinction between program and algorithms. We propose that if there is a problematic area in program verification, it lays into the model construction stage. We suggest an alternative way to account for the distinction between absolute and relative verification by distinguishing the descriptive and prescriptive aspects involved both in program verification and in mathematical proof.

The program verification debate began several decades ago. Already in the fifties McCarthy suggested that debugging should be replaced by formal proof. Furthermore, he claimed, these proofs could be checked by computational tools ([McC62], cited by Colburn p. 7). However, it was the article by De Millo, Lipton and Perlis which in 1979 determined the agenda and the problems which would be central to the formal verification debate. In that work, a distinction was made between proof and demonstration, considering the first as an incomplete draft of a logical demonstration. Both, proof and demonstration, share their aim: to increase the confidence in the correction of a theorem, but whereas a demonstration should in principle complete all the logical steps to ensure the theorem, a proof only represents "one step towards reliability". The differences between mathematics and computational science appear in the practices of each community.

According to Fetzer, the difference between proving a theorem- to "demonstrate" in de Millo's terms - and to verify a program is not the social practice but the presence or absence of causal significance ([Fet88] p. 1061). When there is a physical machine involved then causal factors seems unavoidable. Hence, the program verification as a general method is considered by Fetzer as theoretically unattainable.

Fetzer supposes that there is a sharp and qualitative distinction among program and algorithm -as causal process and logical structures- But it doesn't seem that this distinction can be made in these terms. The argument of Fetzer supposes that if an algorithm is executed in a physical machine, then it falls under causal relationships and therefore they cannot be considered only logical relationships. Therefore, we cannot pursue a complete verification of a 'program'.

But, if this is the correct structure of the Fetzer's argument, then we cannot make proofs in mathematics either. Some kind of physical medium is needed when a mathematical proof is constructed with pen and paper. There is not a qualitative difference between the physical medium used by a program in a concrete machine and the physical medium used by a mathematician making a proof. The same pattern can be regarded if we consider a mathematician making a proof only with his mind - a 'material brain' is also needed. Therefore, the general pattern of Fetzer's argument seems to be misguided, specially the sharp distinction between programs and algorithms.

Futhermore, in Fetzer's description, the relation between algorithms and programs remains obscure. The operation of testing is then the only way to understand what the program does, and this can be checked against its specification (which of course is formal but it can be understood as describing states of the concrete machine) (see figure 1).

This scheme leaves many unsolved problems. Considering a program as a causal process, a test would be only meaningful for this particular machine and for this particular execution. This contrasts with empirical science (or with applied mathematics) where a failed experiment is considered a refutation which affects the theory as a whole, or at least a part of it.

This may be a way out of Fetzer categorial error, a misbehaving program with respect to the specification (and formally verified) could be considered as a refutation of the theory of programming. This theory is the one used to perform the formal verification itself. The prescriptive character of computing science implies in

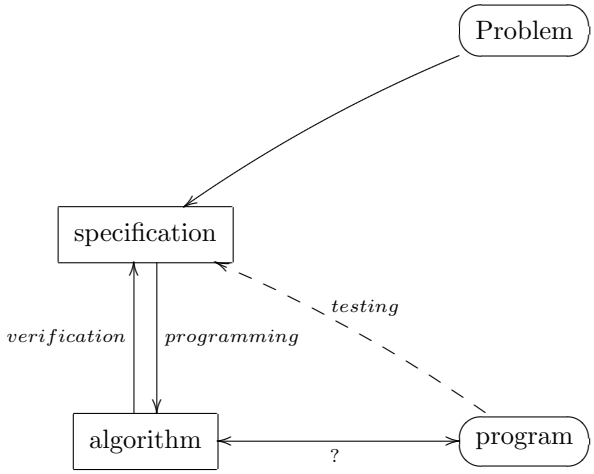

Figure 1: Programs as causal systems

such a situation that if the theory of programming used is consistent and the models supporting it are adequate (whatever this means for the designer), then what is wrong and should be redesigned is the machine or the compiler implementing the programming language.

We believe that algorithm and program are just two aspects of the same thing (let's call it 'program'). The program is a formal object which can prescribe certain behaviors in certain machines. These behaviors can be specified formally (mostly as an input/output relation) and one can mathematically verify this fact. According to de Millo et al. the testing operation against the specification is still needed since verifications are tiresome and not practically feasible. But the information that any test produces is only a constructive proof (in a somewhat unusual medium) that one execution belongs to the specified behaviors for this program [Bor06]. If the formal verification can be constructed, it obviously would imply all the knowledge that all the tests (perhaps an infinite number) can produce. In any case, the only debate here is about practical matters.

Instead, testing may be seen as a step towards bridging the gap between the formal and the real world, comparing what has been modeled with the necessary informal original problem. Actually, this sort of testing can be replaced by a validation process, which can be performed earlier directly from the specification. The triangle with verification, validation and testing in figure 2 should commute.

These last considerations are consequences of a misidentification of verification issues made by Fetzer. This author sees a problem in program verification - uncertainty produced by causal process- and a strategy in testing specifications. The problem and the strategy are originated in his attempt to make a sharp and qualitative distinction between programs and algorithms. And this can be considered a categorial mistake.

The gap between the 'real world' and the computer programs is acknowledged by most computing scientists and software engineers. We tried to show that this gap appears in the modeling stage and is not different than the one in mathematics, it is only easier to see it in computing science given the widespread use of computer programs in many non-mathematical areas.

In this paper we tried to show a way to answer Fetzer's problem, which is founded in the supposition that there is a sharp and characterizable distinction between programs -as causal process- and algorithms -as logical structures-. As a

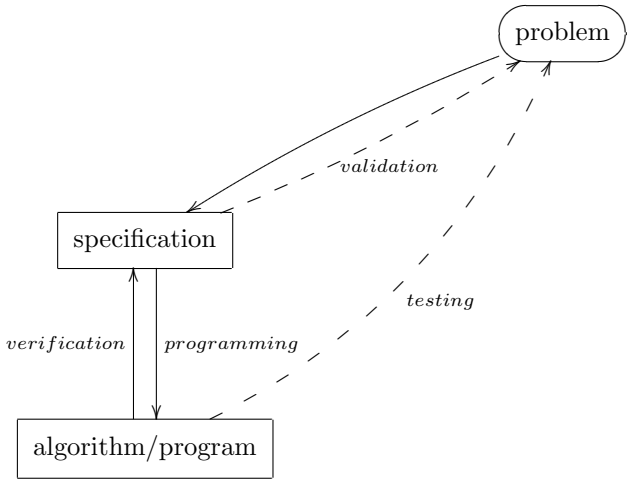

Figure 2: Programs as formal systems

consequence, we would not be able to formally verify programs. We suggest that if this is true, then we could not make mathematical proofs either. We also suggest that Fetzer's proposal leaves many unsoved problems.

We propose, instead of a qualitative distinction between algorithms and programs, that there are logical and causal aspects in programs, which explains the possibility of formal verification but also the possibility of material errors.

References

[Bor06] Richard Bornat. Is computer science science? April 2006.

[DMLP79] Richard A. De Millo, Richard J. Lipton, and Alan J. Perlis. Social processes and proofs of theorems and programs. *Commun. ACM*, 22(5):271–280, May 1979.

[Ede07] Amnon H. Eden. Three paradigms of computer science. *Minds Mach.*, 17(2):135–167, 2007.

[Fet88] James H. Fetzer. Program verification: the very idea. *Commun. ACM*, 31(9):1048–1063, 1988.

[Hoa86] C.A.R. Hoare. Mathematics of programming. *Byte*, pages 115–149, Aug 1986.

[McC62] J McCarthy. Towards a mathematical science of computation. In *Proceedings of IFIP Congress*, pages 21–28, North-Holland, 1962.