The Proof Is in the Process: A Preamble for a Philosophy of Computer-Assisted Mathematics

Liesbeth De Mol

> *Mechanization tends to emphasize practice rather than theory, deeds rather than words, explicit answers rather than existence statements, definitions that are formalized rather than behavioristic, local rather than global phenomena, the limited rather than the infinite, the concrete rather than the abstract, and one could almost say, the scientific rather than the artistic.*
>
> Lehmer (1966)

1 Introduction

Is the computer really affecting mathematics in some fundamental way? Despite the historical connection between mathematics and computers, research within philosophy, history and sociology of mathematics on this question has remained relatively limited.

The main philosophical issues discussed within this context are mostly related to the challenge posed by computer-assisted mathematics to more traditional accounts within the philosophy of mathematics, accounts which view mathematics as an a priori, non-empirical and purely deductive science that generates absolute knowledge through the progressive accumulation of theorems. Computer-assisted proofs of important theorems like the four color theorem by Appel and Haken or the use of "computer experiments" to e.g. give support to important mathematical conjectures *seem* to challenge the very idea of an infallible and a priori mathematics. In this sense, studies of CaM fit in well with the growing emphasis in recent years on

L. De Mol (✉)
Centre for Logic and Philosophy of Science, University of Ghent, Blandijnberg 2, 9000 Ghent, Belgium
e-mail: Elizabeth.DeMol@UGent.be

M.C. Galavotti et al. (eds.), *New Directions in the Philosophy of Science*, The Philosophy of Science in a European Perspective 5, DOI 10.1007/978-3-319-04382-1_2,

mathematical practices.[1] However, not all authors agree on the role of the computer here. In fact it has been argued before that, if experiments exist at all in mathematics, the computer is not (Baker 2008, p. 343) "*an essential feature of experimental mathematics. There is experimental mathematics that makes no use of computers*."

In Avigad (2008) it is argued that some of the typical questions within the philosophical literature on computer-assisted mathematics are too vague. Examples of such questions are (*id*., pp. 3–4):

- In what sense do calculations and simulations provide "evidence" for mathematical hypotheses? Is it rational to act on such evidence?
- Does formal verification yield absolute, or near absolute, certainty? Is it worth the effort?

Instead such questions should be formulated "*in such a way that it is clear what types of analytic methods can have a bearing on the answers*." The task of the philosopher is then to study how these "*pre-theoretic [questions] push us to extent the traditional philosophy of mathematics in two ways: first, to develop theories of mathematical evidence, and second to develop theories of mathematical understanding*" (*id*., p. 5). Hence, we should study such pre-theoretic questions in their proper philosophical context. Furthermore, since "*none of the core issues are specific to the use of the computer* per se [. . .] *issues regarding the use of computers in mathematics are best understood in a broader epistemological context*" Avigad draws two important methodological conclusions from this:

> Ask not what the use of computers in mathematics can do for philosophy; ask what philosophy can do for the use of computers in mathematics [. . .]
>
> What we need now is *not* a philosophy of computers in mathematics; what we need is simply a better philosophy of mathematics.

This paper nicely sums up the general tenor of some of the recent philosophical literature on computer-assisted mathematics: the object under study are issues within the philosophy of mathematics which already have a tradition and, even though the computer raises some questions that challenge more traditional accounts of philosophy of mathematics, these issues are not really essential to the use of the computer.

Even though this approach of studying CaM in a broader philosophical framework is valuable, its insistence on viewing computer-assisted mathematics as something which doesn't really change anything fundamental and merely serves existing debates, runs the risk of underestimating the actual effect on practices of CaM.

A complementary approach which does take the practice of computer-assisted mathematics more seriously seems necessary in order to get a more balanced account of the impact of the computer on (the philosophy of) mathematics. This

[1] See for example van Kerkhove and van Bendegem (2008).

has already been argued to some extent by van Kerkhove and van Bendegem (2008) where it is stated that we *should* account for the practices underpinning formal proofs, including the use of experimental methods (*id*., p. 434):

> [I]t is clear that already today mathematicians rely on computers to warrant mathematical results, and work with conjectures that are only probable to a certain degree. Every so often, we get a glimpse of what is happening back stage, but what seems to be really required is not merely the idea that the front can only work if the whole of the theatre is taken into account, but also that, in order to understand what is happening front stage, an insight and understanding of the whole is required. If not, a deus ex machina will be permanently needed.

But what does it mean to take the machine more seriously? What does it mean to give an account of "the whole theatre"? Several approaches are possible but the one I propose is one which includes a study of the technical details underpinning a practice and, as such, is bottom-up. Within this approach the computer is regarded as a real medium in the sense of people like Friedrich Kittler and Martin Carlé (Carlé and Georgaki 2013):

> The entire impact of a technically informed media theory, from matters of the vowel alphabet all the way to the realm of digital signal processing, brings about one insight: that far more than ideas, it is the 'instrumentality' of thought or the means of communication which establish a dominant regime of knowledge, thus shaping historical reality and its associate notion of truth. Media are no tools. Far more than 'things at our disposal' they constitute the interaction of thinking and perception–mainly unconsciously.

An implication of this point of view is that our mathematical knowledge is really shaped by the machine. The problems that result from its usage *must* thus be regarded as specific to the use of the computer per se. More concretely this view results in a methodology that does not shy away from the "gory" details of the (history of) computer-assisted mathematics and takes the conditions, imposed by the computer on mathematics, more seriously. On the basis of an extensive analysis of CaM, the purpose of this approach is to detect which issues *are* inherent to the use of the computer and, on their basis, to detect how the practice of mathematics is or is not affected by the computer. Such an approach is sensitive to historical fluctuations and does not aim at providing a once-and-for all given answer to the question of the impact of the computer on mathematics.

2 Human-Computer Interactions, Time-Sensitivity and Internalization

In what follows I will focus on experimental mathematics and will thus not consider issues like on-line communities of collaborating mathematicians, the impact of type-setting software on mathematics, etc.

Experimental mathematics is understood here in the sense of number theorist and computer pioneer Derrick H. Lehmer. Lehmer identifies two "schools of thought" in mathematics (Lehmer 1966, p. 745):

> The most popular school now-a-days favors the extension of existing methods of proof to more general situations. This procedure tends to weaken hypothesis rather than to strengthen conclusions. It favors the proliferation of existence theorems and is psychologically comforting in that one is less likely to run across theorems one cannot prove. Under this regime mathematics would become an expanding universe of generality and abstraction, spreading out over a multi-dimensional featureless landscape in which every stone becomes a nugget by definition. Fortunately, there is a second school of thought. This school favors *exploration* [m.i.] as a means of discovery. [B]y more or less elaborate expeditions into the dark mathematical world one sometimes glimpses outlines of what appear to be mountains and one tries to beat a new path. [N]ew methods, not old ones are needed, but are wanting. Besides the frequent lack of success, the exploration procedure has other difficulties. One of these is distraction. One can find a small world of its own under every overturned stone.

For Lehmer it is exactly this possibility of exploration that opens up the path of "*mathematics [as] an experimental science*".

In my previous research I made several detailed case-studies throughout the history of computer-assisted "experimental mathematics" to understand on a more concrete basis the impact of the computer on mathematics. These studies show very clearly the significance of technological advances in computer science (hardware, software and theoretical) for the way experiments are set-up, the types of methods that are developed and the way they are interpreted (see e.g. Bullynck and De Mol 2010; De Mol 2011): in fact, the short history of computer-assisted experimental mathematics itself already underwent important changes due to e.g. increase in computing speed, more efficient read and write operations, developments in programming etc. It are exactly these technological changes that are specific to the use of the computer and allow to trace characteristics of practices of experimental mathematics that come to the fore because of these technological conditions. These characteristics allow to partially explain the increasing popularity of so-called experimental mathematics (see Sect. 2.3).

2.1 Mathematician-Computer Interactions

If there is one characteristic inherent to the use of the computer per se, it is the interaction with the machine. Of course, there is a long history in mathematics of interactions between mathematicians and non-human instruments. The most frequently used is the pen-and-paper method: writing on a piece of paper, a blackboard etc.[2] Figure 1 illustrates the interactive feedback process of such writing practices. Evidently, such interactions are processual – one does not just have one interaction with the piece of paper but many while developing e.g. some idea for a proof or writing a result down to be communicated to the mathematical community.

[2]In this paper I do not consider earlier uses of mechanical devices within mathematics (for instance, Hartree's differential analyzer). These were much less frequently used than digital computers. A comparative study of such devices would be very interesting. It might be that some of the characteristics studied in this paper might also apply to some extent to these earlier devices.

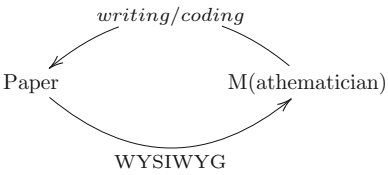

Fig. 1 Scheme of the interactive aspects of mathematical writing practices – first approximation

Within such interaction you write something down. This writing act always involves a certain level of "coding": you use symbols, drawings, abbreviations, plain text etc. This coding practice is historically determined and depends on the goal of the writing act. For instance, mathematical writings in the sixteenth century are very different from mathematical writings in the twenty-first century. Also, writing in notebooks in the process of developing for instance, a good symbolization is very different from writing a textbook. All these writing practices share the property that what you get back from the writing is usually in a WYSIWIG format: this is a term from computer science and stands for What You See is What You Get – it refers to software which uses an interface where you indeed immediately see what you get.[3] When you write or type something on a piece of paper you also immediately get what you write – a stroke is a stroke, a number is a number (at least when your pen or your typewriter are not malfunctioning).

So what happens if we transpose this scheme to interactions with a computer in the context of experimental mathematics? When you are programming a machine to tackle or explore some mathematical problem/object/process, this also involves a process of coding. However, this coding is of a very different nature when compared to "coding" on a piece of paper: whereas the "interpreter" of the mathematical writings on paper is always a human, the coding on a machine has also a non-human interpreter: the machine which executes the code. As a consequence, the coding requires a "language" that is somehow also understandable by a machine. Such intermediary language is called an interface.[4] A mathematician-computer interaction thus involves at least three components: the human, the part of the computer that processes the code and the interface. This results in the feedback scheme of Fig. 2.

This scheme produces several stages of an interaction. First of all, there is the preparatory mathematical stage which involves two substages: first, the understanding of the problem as a computational problem that can be tackled with a machine (Ia). Secondly, there is the translation of the problem into algorithms (Ib). A second stage is the translation of the algorithms to an interface – this is the actual

[3] An example of such software is Office Word where you have a direct visual of the lay-out of your text (e.g. a word in *italics* looks like a word in *italics* on your screen).

[4] Note that the use of the term "language" is a bit tricky here from a historical perspective, hence the use of double quotes. Very early machines like ENIAC did not have such intermediary language: programs had to be set up by physically cabling the machine. In this context the components of the machine, their switches and the cables used to connect them constitute the interface.

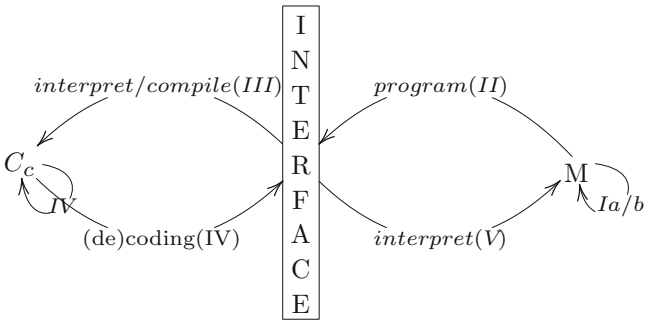

Fig. 2 Scheme of mathematician-computer interactions – first approximation

programming stage and involves the implementation of the problem. Thirdly, there is the interpretation or compilation of the program into machine code and ultimately electronic pulses (Stage III) which can then be executed resulting in an output which has a form specified by the program, e.g., a visualization, a printed (punched) table etc (Stage IV). Note that the interpretation/compilation phase in the modern sense of the word does not really apply for early digital computers. However, there are also certainly processes of translation at work in this context.[5] Finally, there is the interpretation or use of the output by the mathematician which can result in a new programming cycle (Stage V). Figure 2, however, is still a serious simplification of an actual mathematician-computer interaction. When a mathematician is using a computer to do (experimental) mathematics, the interaction is always one that develops over time and in fact involves many sequences of different interactions. For instance, there is always a process of debugging. Furthermore, it is not unusual that several different interfaces are used: the programming interface itself (some text editor like Emacs coupled with an interpreter or compiler), the interface used for the output, a debugger etc. On top of this, the communicative process between the computer and the mathematician is influenced by the communication channels themselves. This was already observed in a different context by Benoît Mandelbrot who refined Shannon's theory of communication by interpreting a communication between a sender and receiver as a game played against Nature (Mandelbrot 1953). These considerations result in more complicated feedback schemes such as the one shown in Fig. 3.

Even though this is still a simplification, this scheme indicates how complicated mathematician-machine interactions in the context of experimental mathematics actually are, involving many different stages affected by the physical and biological conditions imposed by the machine, the interface and the human (Nature). This is a very different kind of interaction when compared with that of Fig. 1.

[5]For instance, in the wiring of conditionals for ENIAC. See Bullynck and De Mol (2010) for more details.

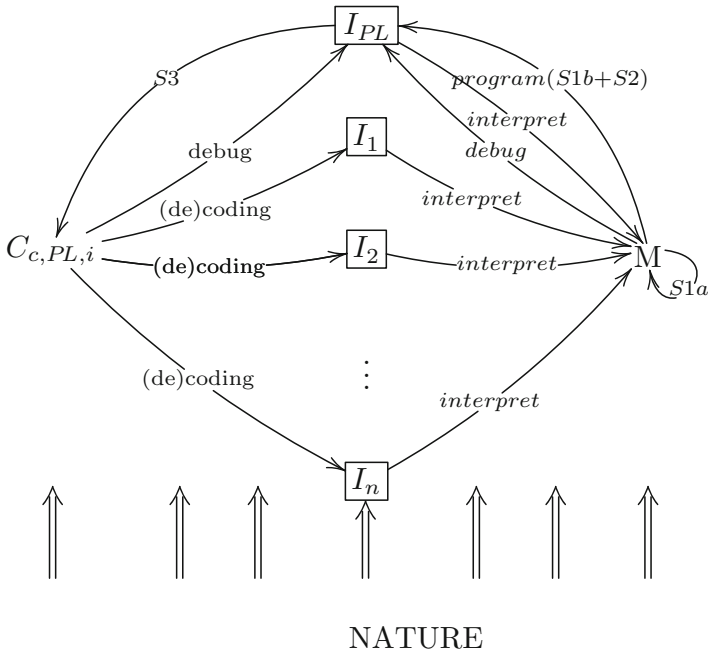

Fig. 3 Scheme of mathematician-computer interactions – second approximation

Of course, one can object against this comparison that the difference lies in the fact that the scheme of Fig. 1 does not include the actor who "reads" and, possibly, "acts upon" what is written on the piece of paper. Hence, the comparison is misleading. Indeed, the piece of paper itself is but a passive receiver of a message which can then be passed on to a second actor: the mathematician him/herself who is writing e.g. in his/her notebooks or a group of mathematicians who are reading what is written on a blackboard (without delay) or in a journal paper (with delay). However, if we were to include this second actor in Fig. 1, we would be comparing interactions between humans and non-humans with interactions amongst humans.

Such comparison between mathematician-computer interactions (MCI) and non-oral interactions between mathematicians requires that we make a detailed study of how the different aspects of MCI are affected by the fact that a non-human is involved. One telling example in this context is the stage of algorithmization: this is shaped by the fact that the algorithm is not meant for human but for machine use, the machine which is assumed to be fundamentally different from humans with its own particular "talents" to tackle a given problem.[6] Throughout the history of CaM this

[6]To put it roughly, a machine has an extremely big memory (at least nowadays) and is much more faster. Moreover it can more easily deal with certain logical complexities. A human on the other

has resulted in the development of entirely new types of algorithms (for instance, the development of pseudo-random number generators) and the transformation of existing human algorithms to machine algorithms.

One early example of this, studied in detail in Bullynck and De Mol (2010), involves the ENIAC, one of the first electronic and general-purpose machines. During a labour day weekend in 1947 Derrick H. Lehmer, his wife Emma and their children spent their time with ENIAC to compute exponents e of 2 mod p, viz. the smallest value of e such that $2^e \equiv 1 \bmod p$. It was a known fact that Fermat's little theorem could be used as a primality test. If for a given number b, $2^b \equiv 2 \bmod b$ than b is with high probability a prime number. Unfortunately, an infinite set of exceptions to this primality test exists. A table of exponents can be used to compute such exceptions. Before, Lehmer had been using Kraïtchik's tables. These tables, however, only extended to 300,000, and contained rather a lot of errors. As a result of his ENIAC computation Lehmer published a list of errors to Kraïtchik's tables and a list of factors of $2^n \pm 1$. Several different subroutines needed to be implemented on the machine including the so-called Exponent routine. Now, this exponent routine is very different from the procedure a human being would follow. A human computer would calculate the exponents e more or less in the following way. First, he would take an existing prime table to select the next prime p. Then he would calculate powers of 2 and reduce them modulo p, though not all powers, but only those that are divisors of $p-1$. This is because a number-theorist knows that if there is an exponent e ($< p-1$) of 2 so that $2^e \equiv 1 \bmod p$ (p prime), than e is either a divisor of or equal to $p-1$. He might also make use of already existing tables of exponents. The ENIAC, in contrast, "*was instructed to take an 'idiot' approach*" (Lehmer 1974, p. 4). First of all, the machine needs a list of prime numbers. Of course, this list of primes could be feeded to the machine by means of punched cards, but, since this is a mechanical procedure, this would significantly slow down the computational process. Hence, it was decided not to use an existing prime table but to let the machine compute its own next value of p as it was needed.[7] The next step was to calculate the powers of 2 reduced modulo p (p being a prime) to compute exponents as follows (Lehmer 1974, pp. 4–5):

> In contrast, the ENIAC was instructed to take an 'idiot' approach, based directly on the definition of e, namely, to compute
>
> $$2^n \equiv \Gamma_n (\bmod p), n = 1, 2, \ldots.$$
>
> until the value 1 appears or until $n = 2001$, whichever happens first. Of course, the procedure was done recursively by the algorithm:
>
> $$\Gamma_1 = 2, \Gamma_{n+1} = \begin{cases} \Gamma_n + \Gamma_n & \text{if } \Gamma_n + \Gamma_n < p \\ \Gamma_n + \Gamma_n - p & \text{otherwise} \end{cases}$$

hand can rely on his/her background knowledge which he/she understands and is able to exhibit human creativity.

[7] This was done by implementing a prime sieve. See Bullynck and De Mol (2010) for more details.

> Only in the second case can Γ_{n+1} be equal to 1. Hence this delicate exponential question in finding $e(p)$ can be handled with only one addition, subtraction, and discrimination at a time cost, practically independent of p, of about 2 seconds per prime. This is less time than it takes to copy down the value of p and in those days this was sensational.

As this example shows, from the very early days of computing, it was necessary to completely transform human methods for tackling a given problem. Viz., the problem needs to be analyzed also from the machine's eye. A simple translation of the human methods would not only result in an extremely slow computation but would also require the internalization into the machine of knowledge it does not really need in order to have an efficient algorithm.

As this short analysis of MCI shows, the digital computer introduces a new type of interaction at work in mathematical practice, a practice that affects the knowledge that results from such interactions. These interactions seem to have more in common with interactions between human mathematicians than with interactions between a mathematician and his writing devices.[8] As such, we are dealing with a new social situation in mathematics which certainly is in need of further analysis.

Whereas this interactive aspect of computer-assisted experimental mathematics can perhaps be regarded as the structure through which practices of experimental mathematics are conducted, there are several other basic characteristics that are part of this interaction and affect the mathematics resulting from it. I will discuss two such characteristics here. The first is the internalization of mathematical tools and knowledge into the machine, the second is the significance of time-based reasoning within such interactions.

2.2 Internalization

If one looks at the short history of computer-assisted mathematics, it is clear that the first examples of so-called computer-assisted experimental mathematics are very different from contemporary experiments. One reason for this is that theoretical and technological advances have affected the way knowledge and skills are distributed between the human and the computer: increases in speed and memory, advances in interfaces, and advances in the "art of programming"[9] have resulted in an increasing internalization of mathematical tools and knowledge into the computer itself. There are two related aspects to such internalization: storage of information in the machine

[8] In this context, it is not surprising that mathematicians who have embraced the computer in their work have insisted on the idea of mathematician-machine collaborations. One interesting example in this context is Doron Zeilberger who has several papers with a certain Shalosh B. Ekhad as his co-author, who is Zeilberger's computer (see http://www.math.rutgers.edu/~zeilberg/pj.html).

[9] This refers to the so-called bible for programmers, viz. Donald Knuth's volumes on *The art of computer programming*.

and algorithmization. Advances in these two aspects of internalization have affected the nature of the interaction between mathematicians and computers.

During the early years of digital computing, there were two major bottlenecks. The first was the programming bottleneck, viz. the fact that it took too much time to set-up a program on a machine because there was no such thing as a programming language (De Mol et al. in press). The second problem was the memory bottleneck: the early machines did not have a large electronic memory. This implies that one could not take advantage of the electronic speed of the machine if one implemented a procedure which needed a lot of intermediary data during the computation itself. One consequence of this was the steady replacement of such tables of data by algorithms – if possible, values were not stored but computed by the machine as they were needed hence resulting in the internalization of data by means of algorithms (see the example of Sect. 2.1). If this was not possible, one had to rely on punched cards which seriously slowed down the computational process.

Another consequence of these two bottlenecks is that the interactive process mostly consisted of clearly separated phases in time. The machine was used mostly for the computational work itself, the calculation. The human took care not only of the programming but also of the exploration or consecutive analysis of the data provided by the machine. For instance, for the Lehmer computation on the ENIAC, the machine was used for the actual computation of the exponents. However, the additional work required to determine on the basis of these exponents the composite numbers was still done by a human. A similar observation can be made for the ENIAC computation of more than 2,000 digits of π and e to explore the statistical distribution of the decimal extension of these numbers. Also in this case, the main computation was done by the machine. However, the statistical analysis was done by humans. Indeed, what one typically sees with these early machines is that the interaction proceeds as follows: first, the program is prepared, then, it is programmed on and executed by the machine. This phase was often followed by one or more "debugging" phases.[10] Finally, the human does the exploration or inspection of the output which might result in a new sequence of preparation, programming, computation, (debugging) and exploration, provided there was enough machine time available. Thus, in the early years of computing, processes of internalization mostly concern the process of algorithmization either to replace the human computational work and to avoid the use of introducing large amounts of data during the computation.

This starts to change with the steady resolvement of the two bottlenecks. The availability of a bigger electronic memory together with advances in programability makes possible the steady internalization of more and more subroutines. This is for instance clear from Grace Hopper's keynote address for the ACM SIGPLAN History of programming languages conference, June 1–3, 1978 (Hopper 1981). She explains, amongst others, how important it was to develop subroutines which

[10]It is interesting to point out that for the ENIAC computation of π and e half of the programming was done to have "*absolute digital accuracy*" (Reitwiesner 1950).

were general enough to be used for a variety of purposes (e.g. a search algorithm that applies to different types) which could then be internalized into the machine (either hard-wired or programmed). One fundamental development in this context mentioned by Hopper is the significance of the machine's ability to write its own program, viz. compiling. This is a precondition for developing programming languages as intermediary languages between the human and the computer. The machine needs the ability to make its own programs in machine code when it is provided with, for instance, the following command in `LISP`, one of the oldest programming languages:

```
(*2 (/ 8 4) (+ 4 6))
```

This possibility of the machine writing its own programs was, initially, met with great skepticism (*id.*, p. 9):

> Of course, at that time the Establishment promptly told us [...] that a computer could not write a program; it was totally impossible; that all that computers could do was arithmetic, and that it couldn't write programs; that it had none of the imagination and dexterity of a human being. I kept trying to explain that we were wrapping up the human being's dexterity in the program that he wrote [...] and that of course we could make a computer do these things so long as they were completely defined.

This is indicative of the transition from using computers as mere calculating aids to machines which can basically do anything an abstract Turing machine can do.[11] This relates directly to processes of internalization: the fact that the computer can do more and more and that this is being understood by humans, goes hand-in-hand with more and more subroutines being internalized into the machine. These subroutines are no longer restricted to the "pure" calculation of "raw data": they are used to visualize data, to statistically analyze data, to inspect data searching e.g. for patterns etc. In this sense, what was in the 1940s and 1950s the human's dexterity is now considered as the machine's dexterity. This development however was possible not only because of increases in programmability but also because technological advances resulted in an exponential increase in the speed by which data can be read or written in the electronic memory, both locally on individual computers as well as globally in networks of computers and humans. Indeed, it makes no sense to have a statistical tool internalized into the computer to deal with billions of data if these data themselves cannot somehow be stored internally into the machine or network of machines.

Nowadays, there exist huge libraries of subroutines not only in programming languages but also in software packages like Maple and Mathematica where "tools of dexterity" can simply be called by their name without the mathematician having

[11] From Hopper's quote one is tempted to conclude that she was not aware of developments in theoretical computer science: it was known since 1936 that there are well-defined problems that cannot be computed, provided one accepts Turing's thesis viz. that anything that is computable is also computable by a Turing machine (see Daylight (2012) for more details).

to know the complete procedures behind such names. One interesting example in this context is Sloane's on-line encyclopedia of integer sequences (OEIS) which is used by several mathematicians as an explorative tool in their work and has resulted in several mathematical papers (see http://oeis.org/wiki/Works_Citing_OEIS for over 2,000 papers that reference the encyclopedia in their work). The encyclopedia stores over 200,000 integer sequences. One very interesting feature of OEIS is that, if you have some number sequence which is not in the encyclopedia and for which you want an explanation, you can mail it to *Superseeker*. This is an algorithm which:

> tries very hard to find an explanation for a number sequence [using] an extensive library of programs that tries a great many things [...] Some programs try to find a formula or recurrence or rule that directly explains the sequence. [...] Other programs apply over 120 different transformations to the given sequence to see if any transformed sequence matches a sequence in the OEIS.[12]

This simple example indicates how processes of increased internalization affect the interaction between the mathematicians and the computer. Whereas in the early years of digital computing the division of labor was very clearly separated into the calculatory work, done by the machine, and the more "intelligent" work done by the human, this division becomes more and more blurred as more sophisticated techniques and more data are internalized into the machine. Nowadays, the machine does part of the programming, part of the inspection, etc. This changes the interaction more and more into a mathematician-machine *collaboration* or, as it has been described before by people like Licklider, a symbiosis:

> Computing machines can do readily, well, and rapidly many things that are difficult or impossible for man, and men can do readily and well, though not rapidly, many things that are difficult or impossible for computers. That suggests that a symbiotic cooperation, if successful in integrating the positive characteristics of men and computers, would be of great value.

However, the fact that more and more information and algorithms are internalized into the machine often means that they are hidden from, inaccessible to or unsurveyable for the (community of) mathematician(s). As such, this situation of increased internalization gives rise to a wide variety of new problems: how can we understand a result if part of it is hidden inside the machine? Can we trust results from the machine? Can we trust the conclusions drawn by fellow mathematicians on the basis of their experiments without having full access to the complete code and data? What does it mean to patent an algorithm? What does it mean for the community of mathematicians that they are using software packages that are not open source and which imply that it is impossible to know all the methods one is using to attain a certain result? etc. These problems lie beyond the scope of this paper but they indicate that the increased internalization of mathematical knowledge into the machine not only affects the interaction between machines and mathematicians but also results in several new problems which cannot simply be discarded.

[12] Taken from http://oeis.org/demos.html on April 5, 2013.

2.3 Time and Finite Processes

Internalization of mathematical knowledge is one aspect that results from interactions between mathematicians and computers. Another fundamental feature is the increasing significance of time and processes in mathematics.

From the early beginning onwards the fact of the speed of electronic computing was, besides its programmability, considered by many a computer pioneer as one of its greatest impacts. Hamming, a mathematician, described the effect of the significance of this electronic computing speed as follows (Hamming 1965, pp. 1–2):

> [An] argument that continually arises is that *machines can do nothing that we cannot do ourselves*, though it is admitted that they can do many things faster and more accurately. The statement is true, but also false. It is like the statement that, regarded solely as a form of transportation, modern automobiles and aerolplanes are no different than walking. [A] jet plane is around two orders of magnitude faster than unaided human transportation, while modern computers are around six orders of magnitude faster than hand computation. It is common knowledge that a change by a single order of magnitude may produce fundamentally new effects in most fields of technology; thus the change by six orders of magnitude in computing have produced many fundamentally new effects that are being simply ignored when the statement is made that computers can only do what we could do ourselves if we wished to take the time.

This speed-up in computation time is often underestimated. It is stated that the mere capability of being faster than a human doesn't change anything fundamental since, *in principle*, we can still do what the machine is doing. It is but a quantitative change. But of course, this *in principle* argument is where the catch lies as indicated by Hamming's quote: *in reality* we simply cannot do what the machine is doing. If one is really taking seriously the mathematical practice, then one *must* account for the qualitative changes that are effected by this quantitative change, else, one should neglect all such, basically, technological changes and one would end up exactly where the philosophy of mathematical practice did not want to go, viz. a largely dehistoricized mathematics which is not sensitive to external changes.

This speed-up of computations goes hand-in-hand with the fact that the objects of computers are not the traditional infinitary and stable objects of mathematics, but highly dynamic and finite processes: a computation is something that develops in and takes time. As a consequence, the computation itself can never be completely captured in the mathematical procedure to be computed and it is the task of the programmer to somehow find a way to control the dynamic processes induced by the program he/she writes: one must be able to write a program that will indeed do what we want it to do. This problem was already understood by John von Neumann who explicitly connected it to the time aspect of computations (von Neumann 1948, pp. 2–3):

> [C]ontemplate the prospect of locking twenty people for two years during which they would be steadily performing computations. And you must give them such explicit instructions at the time of incarceration that at the end of two years you could return and obtain the correct result for your lengthy problem! This dramatizes the necessity for high planning, foresight, and consideration of the logical nature of computation. *This integration of logic in the problem is a consequence of the high speed.* [m.i.]

This need for (logical) control over the behavior of the program is highly relevant in the context of computer-assisted proofs like the four-color theorem, and, more broadly, computer-assisted experimental mathematics. Usually one has thousands of lines of code which makes it extremely hard to verify that the code is doing/will do what it should do. This is the reason why it took for instance several years to review the computer-assisted proof of the sphere packing problem by Thomas Hales. It is also the reason why there is a growing need for formal proofs constructed with the help of proof-assistants like `HOL`. However, to have a formal proof one first needs a traditional proof. Furthermore, "*[i]t is a large labor-intensive undertaking to transform a traditional proof into a formal proof.*" (Hales et al. 2010, p. 3). An alternative strategy to increase the confidence in such computer-assisted results is corroboration. This was for instance proposed by Brady who proved a certain result in theoretical computer science with the help of the computer (Brady 1983, p. 662).

The fact that one needs to deal with highly dynamical processes also very often implies the irreversibility of such processes (Margenstern 2012, p. 645):

> Let us note that in our discrete time of computations, time is irreversible: it is very often extremely difficult to run an algorithm backward. At the highest level of generality it is impossible.

Such irreversibility introduces the arrow of time in the processes studied by means of the computer and, as a consequence, also in those aspects of mathematics that are studied with the help of the machine. In fact, it is this irreversibility in computational processes that has given rise to fundamental problems that resulted in new mathematical developments, for instance, the study of dynamical systems like the quadratic iterator ($f(x_i) = ax_{i_1}(1-x_{i-1})$). Such studies historically originate in the problem of error propagation during computations which became an important problem with electronic computing: since one squeezes thousands of computational steps into a feasible amount of time combined with the fact that computers are finite machines, errors resulting from truncation become highly problematic (see e.g. von Neumann 1948, pp. 3–4).

The irreversibility of computational is also reflected in the languages used to write programs. The most basic example of this is the assignment usually written as[13]:

```
var := expr
```

A simple example of this is:

```
x := x + 1
```

As explained by Margenstern (2012, p. 645), "*the notation* := *explicitly indicates that what is on the left-hand side is not the same as what is on the right-hand side,*

[13]Note that assignment is a typical feature of imperative languages. It is discouraged and sometimes even forbidden in functional languages.

and that there is a process, a consequence of which, after some time, is what we call an [assignment]." Programming languages are full of these kind of notations and, as such, introduce a notation which incorporates the processual character of computation into computer-assisted mathematics.

Such highly dynamic and often irreversible processes also mostly have the property of being unpredictable both theoretically and practically speaking. Hamming describes this unpredictability as follows (Hamming 1965, p. 2):

> One often hears the remark that *computers can only do what they are told to do*. True, but that is like saying that, insofar as mathematics is deductive, once the postulates are given all the rest is trivial. [T]he truth is that in moderately complex situations, such as the postulates of geometry or a complicated program for a computer, it is not possible on a practical level to foresee all of the consequences. Indeed, there is a known theorem that there can be no program which will analyze a general program to tell how long it will run on a machine without actually running the program.

The speed of the machine combined with the theoretical problem that one can often not predict in advance when a program will halt, if at all, implies that we cannot foresee the output. Hence, all one can do is wait and see.

This unpredictability is not just some theoretical problem or property. Indeed, it is in fact this unpredictability that usually brings mathematicians to the computer: because they cannot predict the outcome of a certain computational problem they need to rely on the machine's abilities. This often results in the need for developing local programming strategies which do not always guarantee an outcome. Since in such cases there is often the possibility of infinite programs (for instance, infinite loops) the mathematician has to make certain decisions of when a certain "program" should stop even if it is without outcome. Such decisions are informed guesses based on previous exploratory work. In all of the cases I have studied I found instances of such programs and these are often identified by the mathematicians themselves as "heuristic", "experimental" or "explorative". To give just one example, Brady, when working on his proof mentioned on p. 28 had to program the machine in such a way that it was able to differentiate between different types of infinite loops in the context of Turing machines. Such loop detection is a very difficult task since it involves infinite processes. After several computer-assisted explorations of the behavior of different Turing machines, Brady had identified two types of loops A and B and discovered a property which allowed to *tentatively* but quickly classify a given Turing machine as being a loop of type A, B or an unknown type. This was programmed as a filter called BBFILT and was described by Brady as follows (Brady 1983, p. 662):

> [It] must be remembered that the filtering was a heuristic technique based upon experimental observation.

This is also one of the reasons why he describes his "*proof techniques, embodied in programs [as] entirely heuristic*" (*id.*, p. 647)

Time and processes are an inherent part of MCI: the access to the mathematical results is mediated by highly dynamical processes which introduce the problem of control over and the irreversibility of computational process, reflected in the

language used to communicate with the machine; the mathematician is confronted with the unability to predict what will come and must therefore rely on "heuristic" programming techniques and an external device to get his/her (tentative) answers. These features are part of the practice of computer-assisted experimental mathematics. They not only add an important time dimension to mathematics but even help to partially understand why mathematicians themselves often talk about "experimental mathematics" in this context.

These time-related features are part of the interaction between the mathematician and the computer and, as such, affect it. The fact that one has to wait and see during the sequences of interactions with the machine shifts this interaction further in the direction of a human-human interaction.

Dijkstra, a famous computer pioneer, in discussing the need for a formal semantics of programming languages, once explained the need for human conversation as a means to resolve semantical issues arising from human communication (Dijkstra 1961, p. 8):

> [W]e only know what we have said, when we have seen our listener reacted to it; we only know what the things we are going to say will mean in as far as we can predict his reaction. However, we only know other people up to a (low!) point and in human communication every message is therefore to a high degree a trial, a gamble to see whether the other will understand us as we had hoped. As we do not master the behavior of the other, we badly need in speaking the feed back, known as 'conversation'.

This situation also applies to a certain extent in the context of experimental computer-assisted mathematics: the humanly unpredictable processes of the computer combined with the problem of verifying that the program does what it is supposed to do also introduces uncertainty about the meaning of our own programs.[14] This is exactly why, during such interactions, we cannot simply downplay the replies by the machine as mere results of a computation which we could also have executed *in principle*. Even though we *can* have more control over the behavior of the machine than over that of our fellow human beings, we do not completely master it and as such we need its feedback not only to understand our programs but also to determine our own replies-as-programs to the machine. This is also part of the reason why machine-assisted proofs are presented in a very different manner/style than traditional proofs: since the proof results from a process of MCI in which the work of the computer is not only unsurveyable but also unpredictable and not completely controllable, the proof-as-communicated reflects this processual character of the practice that resulted in the proof. In such published proofs one indeed does not get all the details. But one does get the programs that result in it, a survey of the general structure of the proof, the strategies developed to avoid errors, etc. As such, one sees that the (communicated) proof is not some stable object but a constructive process.

[14]To be clear, Dijkstra would not have agreed on this point: according to him "*[W]e can fully master [. . .] the way in which the computer reacts*" But see in this context the quote by Hamming on p. 29.

3 Discussion

What is the impact of the computer on mathematics? From a philosophical meta-perspective the answer seems to be that nothing fundamentally changes to mathematics itself since, *in principle*, the machine can do nothing that we cannot do ourselves and it merely does what it is told to do. It is admitted that the computer does pose some new challenges for traditional philosophical problems but this merely shows that we are in need of a better philosophy of mathematics that can deal with these challenges. A serious philosophy of computer-assisted mathematics however is considered to be unnecessary.

Even though such views are perfectly arguable from a meta-perspective, they run the risk of underestimating the effects of the computer on mathematics *in reality* and on its philosophy which is itself rooted in the history of mathematics and hence sensitive to change. I am strongly convinced that it would be a missed chance for the philosophy of if it would not even make the effort of investigating more seriously practices of computer-assisted mathematics for their own sake (and less for existing philosophical debates). To this end, I have proposed an approach which takes the machine seriously as a *medium*. Such view implies that we *do* need a philosophy of the computer (in mathematics). Within such an approach, one conducts research from the bottom-up in order to trace down characteristics of computer-assisted practices which are specific to the use of the machine.

In this paper I have discussed three such characteristics: MCI, the steady process of internalization of knowledge and techniques into the machine and the significance of time and processes within computer-assisted practices. One could of course argue against this that one already has internalization of knowledge before the rise of the computer in another form, viz. by way of writing and the printed press. Similarly, one could say that since computations were already important to mathematics before the digital computer this processual nature was already part of mathematics long before the rise of the digital computer. And indeed, the claim of this paper is certainly not that there is some sudden discontinuity from what was before. What I do claim here is that these two further characteristics, as being aspects of the mathematician-computer interaction, are seriously affected by the machine and as such gain a new meaning resulting in an effect on mathematics proper. Viz., the machine has not resulted in an immediate and sudden change, but it is steadily changing features that are inherent to the practice of the mathematician, knowledge transfer, communication, collaboration, mathematical notation, etc. are changing due to the use of the computer.

Two important consequences for mathematics follow from the present discussion and show that we are in need of a better understanding of practices of CaM. Firstly, the computer introduces a new social situation into mathematics: the interaction between digital machines and mathematicians. It is striking that, on the basis of the analyses from Sects. 2.1–2.3, this interaction is shifted into the direction of communication and even collaboration between human mathematicians. Evidently, the two forms of interaction shouldn't be identified because of the active

involvement of a non-human. However, it does show that one cannot simply discard the machine as being just another tool and that further research into comparing these two modes of interaction is necessary. One obvious approach for such a comparison might be to build formal models which allow a more detailed and exact comparison.[15]

Secondly, computer-assisted mathematics explicitly and abundantly introduces time into the practice of the mathematician. John von Neumann, who was very keen on using digital machinery to study problems of applied mathematics, once stated that as mathematics (von Neumann 1947):

> travels far from its empirical source, or still more, if it is a second and third generation only indirectly inspired by ideas coming from 'reality', it is beset with very grave dangers. It becomes more and more purely aestheticizing, more and more purely *l'art pour l'art*. [...] there is a grave danger that the subject will develop along the line of least resistance, that the stream, so far from its source, will separate into a multitude of insignificant branches, and that the discipline will become a disorganized mass of details and complexities. In other words, at a great distance from its empirical source, or after much 'abstract' inbreeding, a mathematical subject is in danger of degeneration [W]henever this stage is reached, the only remedy seems to me to be the rejuvenating return to the source: the reinjection of more or less directly empirical ideas. I am convinced that this was a necessary condition to conserve the freshness and the vitality of the subject and that this will remain equally true in the future.

Far more than reinjecting empirical ideas into mathematics, perhaps the computer is reinjecting time into a discipline that has long been regarded as being above and without time.

References

Allo, P., J.-P. van Bendegem, and B. van Kerkhove. 2013. Mathematical arguments and distributed knowledge. In *The argument of mathematics*, ed. A. Aberdein and I.J. Dove, 339–360. Berlin: Springer.

Avigad, J. 2008. Computers in mathematical inquiry. In *Philosophy and the many faces of science*, ed. P. Mancosu, 302–316. Oxford.

Baker, A. 2008. Experimental mathematics. *Erkenntnis* 68: 331–344.

Brady, A.H. 1983. The determination of the value of Radó's noncomputable function σ for four-state turing machines. *Mathematics of Computation* 40: 647–665.

Bullynck, M., and L. De Mol. 2010. Setting-up early computer programs: D. H. Lehmer's ENIAC computation. *Archive for Mathematical Logic* 49: 123–146.

Carlé, M., and A. Goergaki. 2013. Re-configuring ancient Greek music theory through technology: An adaptive electronic tuning system on a reconstructed ancient Greek barbiton. In *La musique et ses instruments* [Music and its instruments], ed. M. Castellengo and H. Genevois, 331–380. Paris: Editions Delatour.

Daylight, E. 2012. *The dawn of software engineering: From turing to Dijkstra*. Heverlee/Belgium: Lonely Scholar.

[15] See Allo et al. (2013) where epistemic logics are used to study processes of finding proofs by communities of mathematicians.

De Mol, L. 2011. Looking for busy beavers. A socio-philosophical study of a computer-assisted proof. In *Foundations of the formal sciences VII. Bringing together philosophy and sociology of science*, Studies in logic, vol. 32, ed. K. Francois, B. Löwe, Th. Müller, and B. van Kerkhove, 61–90. London: College publications.

De Mol, L., M. Carlé, and M. Bullynck. in press. Haskell before Haskell: An alternative lesson in practical logics of the ENIAC. *Journal of Logic and Computation*. doi:10.1093/logcom/exs072.

Dijkstra, E.W. 1961. On the design of machine independent programming languages. Report MR34, Stichting Mathematisch Centrum, Amsterdam.

Hales, Th., J. Harrison, S. McLaughlin, T. Nipkow, S. Obua, and R. Zumkeller. 2010. A revision of the proof of the Kepler conjecture. *Discrete and Computational Geometry* 44: 1–34.

Hamming, R.W. 1965. Impact of computers. *The American Mathematical Monthly* 72: 1–7.

Hopper, F.M. 1981. Keynote address. In *History of programming languages*, ed. R.L. Wexelblat, 7–20. New York: Academic.

Lehmer, D.H. 1966. Mechanized mathematics. *Bulletin of the American Mathematical Society* 72: 739–750.

Lehmer, D.H. 1974. The influence of computing on mathematical research and education. In *Proceedings of symposia in applied mathematics*, vol. 20, ed. J. Lasalle, 3–12. Providence: American Mathematical Society.

Mandelbrot, B. 1953. *Contribution à la théorie mathématiques des jeux de communication*. Paris: Laboratoires d'électroniques et de physique appliquées.

Margenstern, M. 2012. Comment. In *A computable universe*, ed. H. Zenil, 645–646. Singapore: Worldscientific.

Reitwiesner, G.W. 1950. An ENIAC determination of π and e to more than 2000 decimal places. *Mathematical Tables and Other Aids to Computation* 4: 11–15.

Van Kerkhove, Bart, and Jean Paul van Bendegem. 2008. Pi on earth, or mathematics in the real world. *Erkenntnis* 68(3): 421–435.

von Neumann, J. 1947. The mathematician. In *The works of the mind*, ed. R.B. Heywood, 180–196. Chicago: University of Chicago Press.

von Neumann, J. 1948. Electronic methods of computation. *Bulletin of the American Academy of Arts and Sciences* 1: 2–4.

http://www.springer.com/978-3-319-04381-4

New Directions in the Philosophy of Science

Galavotti, M.C.; Dieks, D.; Gonzalez, W.J.; Hartmann, S.;
Uebel, Th.; Weber, M. (Eds.)

2014, XII, 773 p. 22 illus., Hardcover
ISBN: 978-3-319-04381-4