Some reflections on mathematics and its relation to computer science

Liesbeth De Mol

Some reflections on mathematics and its relation to computer science

Liesbeth De Mol*

"*No paradigm should ever be allowed to dominate education*"

Benoît Mandelbrot, 2002

In [24, p. 325] Knuth recounts the following story:

> [At some time,] I wondered how to calculate the greatest common right divisor of two given matrices. A few days later I happened to be attending a conference where I met the mathematician H.B. Mann, and I felt that he would know how to solve this problem. I asked him, and he did indeed know the correct answer; but it was a mathematician's answer, not a computer scientist's answer! He said, "Let $\mathscr{R}$ be the ring of $n \times n$ integer matrices; in this ring, the sum of two principal left ideals is principal, so let D be such that
>
> $$\mathscr{R}A + \mathscr{R}B = \mathscr{R}D$$
>
> Then D is the greatest common right divisor of A and B." This formula is certainly the simplest possible one, we need only eight symbols to write it down; and it relies on rigorously-proved theorems of mathematical algebra. But from the standpoint of a computer scientist, it is worthless

Knuth used this story to explain how he considered mathematics and computer science to be distinct from each other: it illustrates very clearly the different ways by which mathematicians and computer scientists approach a given problem. Despite these differences in thinking, it is not uncommon for computer

*Centre Nationale de la Recherche Scientifique, UMR Savoirs, Textes Langage, Université de Lille 3. This paper resulted from a talk I gave at Machines, Computations and Universality 2013 in Zürich and I am very much indebted to the organizers and the participants of this conference for a very fruitful discussion. I am particularly grateful to Maurice Margenstern who, since he was a reader of my PhD, has given me several useful advices related to my work and has often motivated me for inquiring further into problems of decidability and undecidability in the context of tag systems, and more generally, for developing my thoughts on experimental mathematics and computer science. Of equal importance is the fact that it is thanks to Maurice that I was able to defend my Ph.D. thesis without academic obstruction.

scientists to stress also the similarities between their discipline and mathematics. For instance, Dijkstra has argued that the methods of programming are mathematical in nature [15] and Wegner has emphasized on several occasions that computer science is "*in part a mathematical discipline*" [39].

These reflections on the relation between mathematics and computer science usually take place in a computer science context since, today, most mathematicians do not really feel the need to define their discipline with respect to another. However, since the significance of the computer is increasing within mathematics, some mathematicians have started to reflect on this issue by considering the question how mathematics is being affected by the use of the computer. Most well-known at this time is probably the work by mathematicians like Borwein who have extensive experience with so-called computer-assisted experimental mathematics and who has argued on multiple occasions that "*[t]he computer is changing the way we are doing mathematics*".

The aim of this paper is to revisit the historically-developed question on the nature of the relation between mathematics and computer science by (mainly) focusing on mathematical practices that involve the use of the computer. This will allow me to highlight *some* aspects of mathematics that are being affected by what I like to call a computer-science way of thinking. By doing so I not only want to place this "way of thinking" into a broader historical context of mathematics but also offer some reflections on the computer science discipline itself.

Considering this relation between mathematics and computer science, it is unavoidable to represent things as if they were black and white whereas in real practice this is only rarely ever true. It is for this reason that I would like to recall here, as a kind of appeal to the reader, the words by Hamming who stated during his Turing award lecture [22]:

> We live in a world of shades of grey, but in order to argue, indeed even to think it is often necessary to dichotomize and say "black" or "white". Of course in doing so we do violence to the truth, but there seems to be no other way to proceed. I trust, therefore, that you will take many of my small distinctions in this light – in a sense, I do not believe them myself, but there seems to be no other simple way of discussing the matter."

1 Mathematical logic, the computer and mathematics

Several decades before computer science was recognized as a discipline, mathematics was already inexorably tied up with what were to become some of the foundational sources for computer science. The papers by mathematicians like Church, Curry, Gödel, Hilbert, Kleene, Post and Turing are nowadays considered as fundamental sources of (theoretical) computer science but resulted in a context of reflections on the foundations of mathematics. Hilbert's for-

malistic and finitist program was one important school of thought within this foundational context.[1] He understood formalism and the method of finite axiomatization as a way to tackle several foundational problems of mathematics, viz., problems about *all* possible assertions in mathematics, most notably, consistency. He believed that by addressing such problems through finitist and formalist means, mathematics could be provided with a firm foundation (See for instance [32]).

One important conviction of Hilbert was that within mathematics, there is no Ignorabimus. Indeed, as Hilbert famously stated during a lecture in 1930 in Königsberg[2] (translated from [23, p. 963]):[3]

> The true reason why Comte could not find an unsolvable problem, lies in my opinion in the assertion that there exists no unsolvable problem. Instead of the stupid Ignorabimus, our solution should be: We must know. We shall know.

Others however were less positive about the prospect of having a mechanical solution to decide any mathematical proposition. As Von Neumann stated in 1927 (quoted from [17]):

> [T]he contemporary practice of mathematics, using as it does heuristic methods, only makes sense because of this undecidability. When the undecidability fails then mathematics, as we now understand it, will cease to exist; in its place there will be a mechanical prescription for deciding whether a given sentence is provable or not.

As we all know now, Hilbert was in fact too optimistic: in 1930-31 Gödel proved the limitations of the method of finite axiomatization and formalization through his incompleteness theorems and in 1936 Church and Turing independently proved the undecidability of the decision problem for first-order logic.[4] These results are often interpreted as the death-knell to Hilbert's finitist and formalist program. They certainly were for his dream of a mathematics without Ignorabimus!

Ironically, it were exactly the different formalist devices and techniques used and/or developed as tools to obtain such impossibility results, that would turn out to be very useful instruments for developing (some of) the theoretical foundations of and tools for the machine that *can* be seen as thé formalist device per se, viz. the computer. The computer is not only finite per definition but its actions are those of the cliché formalist practice, viz. the blind manipulation of meaningless symbols according to some rules. It was exactly for this reason

[1] One other such school is Brouwer's intuitionism which today surely also has an important impact on computer science by means of constructive type theory.

[2] Ironically, Gödel would announce his incompleteness results at the same meeting!

[3] Hilbert repeated his belief in the non-existence of Ignorabimus in mathematics on several occasions.

[4] It is less well-known that Emil Post had already obtained incompleteness and undecidability results in the early 20s in the context of so-called normal systems. These results were later published as [35].

that people like Dijkstra made a plea on multiple occasions for the significance of formalism in the context of computer science [16]:

> The manipulation of uninterpreted formulae is [...] a most familiar operation for the computing scientist: it is the one and only operation computers are very good at. [...] The manipulation of uninterpreted formulae requires unambiguous formalisms [...] Our disappointing experiences with formality should not be interpreted as something being wrong with formality; the experiences were disappointing because we were incompetent amateurs. But this has been changed by our exposure to computing

Today, the idea to use formalist techniques as a tool has become part of the standard practice within computer science. A leading thought is that of controlling problems of unreliability, unpredictability and complexity which frequently occur in the context of computing and is rooted in a belief that, to quote Dijkstra again, "*[m]astery of the reaction of the computer must not only be a theoretical possibility but a real, practical one*" [14]. As such, formal verification, denotational semantics, Chomsky grammars etc are today used within (the development of) programming languages and compilers in order to deal with problems of error, non-termination, security, etc. Also within mathematics, this formal approach is applied: proof assistants like Coq, which help to formally specify and verify mathematical proofs, have been used or are being used to formalize contested mathematical results, like the four-color theorem or the sphere packing problem. Such formalized proofs allow to control and verify that the steps taken by the machine are correctly executed and indeed lead to a given theorem.

However, such practical realizations of formalism(s) are just one aspect of the computer and its surrounding practices. There is also a "non-rigorous" side which relies on experimentation, statistics, etc. to deal with problems of unpredictability, unreliability and complexity on the level of hardware, software, the humans that rely on it and the problems that are studied with it. For instance, in compiler design, experiments have been executed to determine the optimal size of a hash table; within hardware design statistical methods are used to regulate the cache memory; during programming, the debugging and testing of code is often preferred over formal verification, etc. The interplay between the different levels involved in computing is often too complex to permit for a feasible formal method and is thus bound to escape its formal control (practically or theoretically speaking). This more "ugly" side of computing, is, in a certain sense, more in line with von Neumann's preference for a mathematical practice which relies on heuristics.

It is exactly within this historically developed mathematical practice that we find another fundamental connection between mathematics and computer science, viz. computations and algorithms as the means to execute them. Indeed, algorithms and computations have always been a part of mathematics. However, this significance of computation and algorithms within mathematics, evident though it may seem from our contemporary perspective, has not always been properly acknowledged. Indeed, many of us have been educated with a

mathematics that is more about abstract and general structures and theories than about concretely computed objects. The reason for this is that for a long time, a majority of mathematicians simply preferred general and abstract results over particular computation-based results.[5] As is suggested in [10], the separation between "pure" mathematics, on the one hand, and computation-intensive mathematics on the other, became sharper over the course of the 19th century and the beginning of the 20th century. Interestingly enough, Hilbert played an important role in this trend of treating "computation" rather pejoratively: in an influential report on algebraic number theory, known as *Zahlbericht* and published in 1897, Hilbert favors a more conceptual approach over a more algorithmic approach and certainly preferred "pure" ideas over computation (Quoted from [10]):

> I have tried to avoid Kummer's elaborate computational machinery, so that here too Riemann's principle may be realized and the proof completed not by computations but purely by ideas

This idea to obtain results purely by ideas rather than "computational machinery" is still quite popular within mathematics. However, with the rise of the computer one also sees a slow renaissance of computations and algorithms: so-called "experimental" or "explorative" mathematics is becoming more popular, curricula no longer shy away from discrete mathematics and a growing number of mathematicians is focusing on mathematics of computation. Interestingly enough, it is exactly this "style" of mathematics that von Neumann, who himself was for a long time a clear supporter of the formalist school of thought (See e.g. [38]), promoted and practiced in the last 10 to 15 years of his life, viz. a mathematics that is rooted in computational results. In fact, von Neumann understood this possibility which the new computing machines were offering to the mathematician as a way to escape from a mathematics "*in danger of degeneration [after] much "abstract inbreeding"* [36].

As becomes clear through this account, the computer and with it, computer science, incarnates (at least) two different "styles" or "approaches": on one side of the spectrum one finds a more Hilbertian style which we can associate with abstract, elegant, rigorous and formal approaches on computation, on the other side, we find a late-von Neumann style which is usually considered more ugly, computation-intensive and more experimental. Evidently, in the context of computer science, both approaches are strongly connected through the computer and its computations. As such, they cannot be strictly separated from one another. Both can also be traced within the history of mathematics, even though the first seems to express the current more dominant view on mathematics.

Today, a growing community of mathematicians is embracing the computer to advance their work and, with it, the so-conceived less elegant style of doing mathematics. Indeed, despite the fact that the computer *is*, from a certain

[5] This does not necessarily mean that for some period in the history of mathematics, computations and algorithms were no longer used or developed. Rather it means that they were not explicitly a part of the general mathematical discourse.

point of view, a formalist device, its effect on mathematics proper[6] apparently lies exactly in the opposite of being formal. So what exactly is it with the computer which encourages this style of mathematics and how does it relate to more formalist approaches within computer science? More generally, how is the computer affecting mathematics?

2 A number-theorist's point of view

What is the impact of the computer on mathematics? An important source of inspiration for my own work on this question is Derrick H. Lehmer: he was one of the first mathematicians, a number theorist, to use a computer for doing mathematics and he has written on several occasions on the potential of extensive computation for mathematics. He identified two schools of thought within mathematics [31, 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.

Lehmer clearly had a preference for the more "experimental" school of thought: having been raised by Derrick N. Lehmer, also a number theorist and well-known table-maker of prime numbers and factors, he was convinced of the experimental nature of mathematics and especially number theory. As he explains himself: "*My father did many things to make me realize at an early age that mathematics, and especially number theory, is an experimental science*" Such experimental approach usually requires exploration and hence also something to explore. It is thus not surprising that Lehmer – and with him several other mathematicians – regard(ed) the computer as the perfect partner to support the "exploratory-minded". Indeed, the increase of "several orders of magnitude" in speed and memory combined with the capability of the machine to deal with combinatorial

[6] Viz., not a mathematically-oriented computer science

complexities on the executional level of a program, makes it possible "*to explore the terrain that has been staked out so freely and that something worth proving will be discovered in the rapidly expanding universe of mathematics*" [30, p. 146].

But what kind of mathematics is such "explorative" computer-assisted mathematics? Let us first look at a simple example and then relate it to Lehmer's view on this. The example comes from the early history of computer-assisted mathematics and concerns the study of the decimal expansion of π and e on ENIAC, one of the first electronic and (externally) programmable machines.[7] It is rather well-known that one of the applications of ENIAC was the computation of multiplication rates of neutrons in fission devices to estimate the size of the device for triggering fusion in an H-bomb. One of the people involved with this project was John von Neumann. He was convinced that a more experimental approach was the most suitable for studying this problem and it was decided to let ENIAC "numerically simulate", amongst others, the multiplication rates. It was in this context that the now so widely used but rarely questioned Monte Carlo method was introduced by Ulam. Of course, in order to compute with the Monte Carlo method one needs a source of random numbers. The most obvious method at the time was to use numbers generated by some other device (e.g. a roulette wheel) but, since electronic memory was limited at the time, these numbers had to be fed mechanically to the machine and significantly slowed down the computational process. Von Neumann and others started to reflect on the possibility of letting the machine generate its own random numbers resulting in the construction of so-called pseudo-random generators. However, how should one construct an algorithm for generating random numbers? Isn't it paradoxical to generate randomness through deterministic procedures? It is against this background that one should understand the computation of 2,000 digits of π and e by ENIAC: the aim was to investigate the statistical distribution of these numbers.[8] ENIAC was used to compute these digits and then a team of human computers was used to perform the statistical analysis. Also other methods were studied, including von Neumann's middle square method[9] – which was eventually used – and the quadratic iterator ($x_i = ax(1x_{i-1})$) which is now known as one of the paradigmatic cases of chaotic non-linear equations.

This example is a very simple but clear example of explorative mathematics: not only because it explicitly concerns the exploration of the digits of π and e but also because of the wider background: the search and study of pseudo-random generators by exploring several possible instances. This act of exploration assumes several other activities. Amongst others, it necessitates the development

[7] ENIAC in its initial form was not a stored-program computer. Instead one had to manually wire the machine for each computation. Nonetheless, it was Turing complete in the same sense as modern machines are, viz., making abstraction from its memory limitations, it could simulate any Turing machine (amongst others, it had a conditional). For a detailed example of a program that was actually ran on ENIAC see [6].

[8] The exact nature of the distribution of these numbers is still unknown today even though it has been surmised more than once that they are normal.

[9] This method is known to be a very bad random generator in general. It functions as follows: pick a number with n digits, square it, resulting in a number of $2n$ digits, take its n middle digits, square it, etc.

of several special techniques: the development or selection of feasible approximating iterative algorithms that are fitted to the digital ENIAC; techniques to check for possible errors in the computation; etc.

This type of study of the decimal expansion of $\pi$and e was not new but in fact fits into a wider computational and explorative tradition: long before ENIAC, expansions of π and e were computed for several different reasons – as an exploration or test of good approximation algorithms, as a study of whether π is rational are not, etc[10] In other words, even though the local context changes greatly, there is an almost natural bond between this type of explorative mathematics and extensive computation. Since the electronic computer is the machine which opens up the path of extensive computation, its usage in such explorative contexts is certainly not surprising. However, by using the computer, the mathematician also engages with a particular type of thinking summarized by Lehmer as follows [31]:

> I should like to speculate briefly on the overall impact of mechanization upon mathematics of the not too distant future. 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 [...] The computer is the instrument of our observatory, our window to the hard facts of the world of mathematics.

If one were to differentiates computer science from mathematics, the finite vs. the infinite, the concrete vs. the more abstract, etc are indeed the kind of typical differences one immediately comes up with. Even though abstraction from the machine, the development of formal and uniform methods and infinite models are a part of the usual practice in computer science, the object remains limited, discrete, local, concrete and practical. It is perhaps for this reason that, as Knuth explains, computer scientists are more familiar with non-uniformity and case-by-case analyses [27]:

> If I had to put my finger on the greatest difference between mathematicians and computer scientists, I would say that mathematicians have a strong preference for non-uniform rules, coupled with a strong dislike for case-by-case analysis; computer scientists, by contrast, are comfortable and fluent with highly non-uniform structures (like the different operations performed by real computers, or like the various steps in long an complex algorithms).

It is also this type of thinking that fits well with exploration: one needs to define local methods or algorithms that work in the particular context, the methods

[10]It was only in the 18th century that Legendre and Lambert proved the irrationality of π

need to be finite (one cannot "explore" the infinite as such, only its finite approximations) and one often needs to proceed on a case-by-case basis. In other words, what I claim here and what Lehmer claims is that some of the typical features of what one could tentatively call a computer science way of thinking has a common basis with that fraction of the mathematical discourse which is usually associated with exploration and computation. It is thus not surprising that, in view of a supposed increasing significance of the computer, Lehmer makes the following two possible predictions about the future of mathematics [31]:

> It would be a pleasure to predict that, as time goes on, the use of the instrument will become widespread and the nature of mathematics will slowly change from the dangerously unstable fluid art that it is apparently approaching today to a more and more structured and explicit science. There is an alternate prediction. Already we see, instead, a splitting from mathematics of a new branch commonly called computer science, which includes enough technology to frighten away your topologist or functional analyst. Soon disciplinary fences will be erected. It has been said that the invention of photography relieved the graphic artist of his obligation to depict nature and drove him into impressionism and finally to abstraction. This, it seems to me, is apt to be also the future of mathematics.

It is impossible at this point in history to verify which of these two predictions is the most correct one – time will have to tell. However, what we can do is to partially verify the current tendencies within mathematics with respect to computing and exploration.

3 The impact of the computer on mathematics: some quantitative results

In order to find at least a heuristic guide to trace the impact the computer is having on mathematics, I studied the quantitative impact of the computer on mathematics.The method consists in measuring the frequency by which particular terms are being used within the mathematics discipline as represented in databases such as MathSciNet and Zentralblatt. This study is not completed yet, but some initial results at least give some indications of this overall impact. This far, I focused on MathSciNet and three groups/clusters of terms: Group I consists of the words: comput*, calcula*, machine*; group II consists of: experiment*, heurist*, conject*, explor*, inspect* and, finally, group III consists of: Algorith*, program*, automat*, digital*, simulation*. Fig. 1–3 show a plot of the evolution of the frequency of these three groups of terms as used within title or review text of items listed in MathSciNet between 1940–2010.

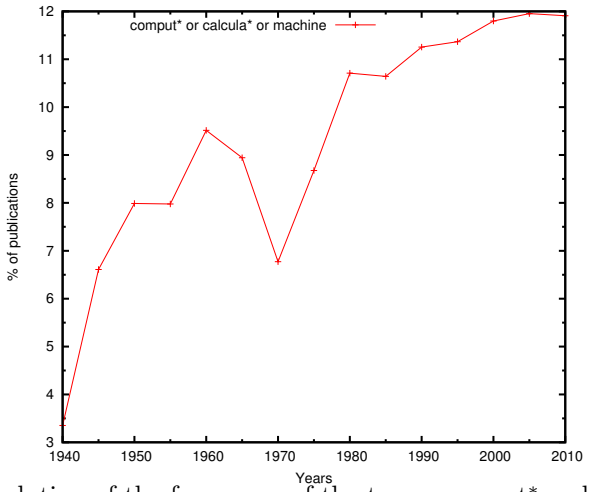

Figure 1: Evolution of the frequency of the terms comput*, calcula*, machine* in MathSciNet 1940–2010

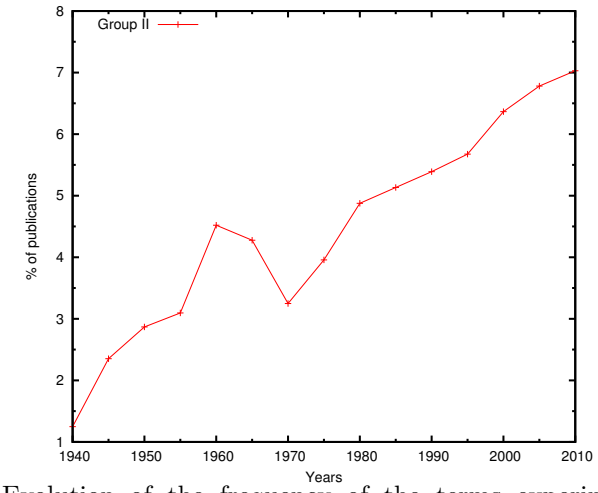

Figure 2: Evolution of the frequency of the terms experiment*, heurist*, conject*, explor*, inspect* in MathSciNet 1940–2010

As is clear from these plots, the significance of these three groups increases as a function of time: by 2010, almost 10% of all items listed in MathSciNet explicitly refer to terms that are connected to computers and computing in title or review text. Also the usage of terms that are often associated with exploration, terms such as experiment, conjecture, etc, show a steady increase with about 7% in 2010. But most interestingly is the evolution of group III with no less than +/- 13% of the publications included in MathSciNet containing one of the terms of group III in its title or abstract by 2010. A more detailed

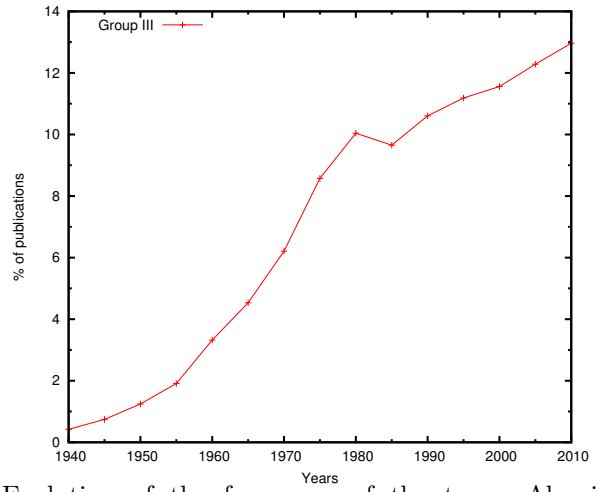

Figure 3: Evolution of the frequency of the terms Algorith*, program*, automat*, digital*, simulation* in MathSciNet 1940–2010

analysis of the results also shows that within each of the three groups there are always one or two terms that take up most of the percentages. In group I, the terms starting with comput* and calcula* are the most dominant, with comput* steadily taking over from calcula*. For group II, the dominant terms are conjecture* and experiment*. Finally, for group II, it are algorith* and program* that are very dominant (by 2010, they take up about 9%) though one can observe a steady increase also in the usage of simulation* starting around 1980.

So what do these results indicate? Even though further analysis is required combined with an extension to other databases, the results at the very least indicate that one cannot neglect the impact of the computer on mathematics by arguing that there is a neglectable fraction of mathematicians interested in computing: both groups I and III, which concern a terminology that is immediately related to computer science, show a significant increase since the early years of digital general-purpose computing. This increase cannot be explained by the development of the computer science discipline alone: a more detailed study of the distribution of group I over the disciplines, using the MSC 2010 classification of disciplines, shows that for the computer science related disciplines like MSC 65 (Numerical Analysis) or MSC 68 (computer science) there is in fact a decrease in the significance of Group I which starts already around 1950 and then stabilizes around 1970. This decrease can be explained by a complementary increase in the usage of the terms algorith* and program* as is shown in Fig. 4. Whereas the results for Groups I and III are explicitly related to the impact of the computer on mathematics, the results for Group II are not. Nonetheless it cannot be neglected that the usage of terms associated with "experimental" mathematics shows a steady increase which parallels the one for Group I.

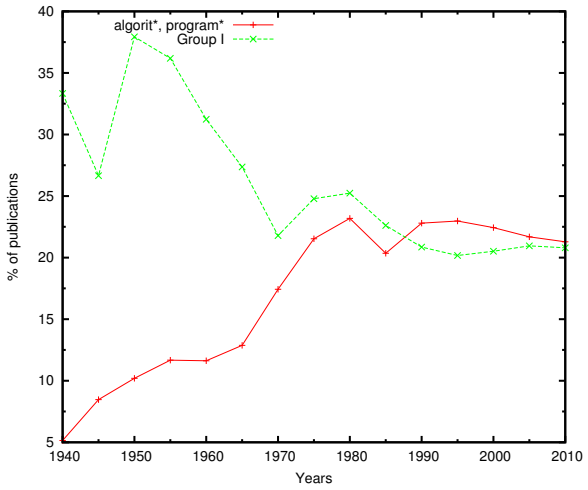

Figure 4: Evolution of the frequency of the terms of group I versus that of algorith* and program* in the disciplines 65* (numerical analysis), 68* (computer science), 90* (operations research, mathematical programming), 93* (systems theory; control) and 94* (information and communication; circuits)

These results are what they are: a mere quantitative argument showing that there is a significant impact of the computer on mathematics. For now they do serve the purpose of testing Lehmer's two alternative predictions in our time: even though it is indeed true that, in the meantime, disciplinary fences have been built there is a clear indication that computation, calculation, algorithms, programs, conjectures and experimentation have become much more important over the years also within mathematics. However, these quantitative results do not allow us to *directly* tackle the question of the qualitative impact of the computer on mathematics but can at best serve as a heuristic guide to detect more carefully the impact of the computer on mathematics.

4 Computer-assisted explorative mathematics: Characteristics and problems

How is the computer changing mathematics, if at all? This far I have looked at this question mostly from the perspective of mathematics itself rather than from the computer, focusing first on the particular views of Derrick H. Lehmer and then at some more global developments we can detect within mathematics. But what is it exactly that the computer brings to mathematics besides extensive computational results to be explored? What particular characteristics are there to the computer that help to understand or clarify its (potential) effect on mathematics? In [13] I discuss three features which I consider as characteristic to computer-assisted explorative mathematics: human-computer interaction;

the significance of time and processes and the internalization of mathematics into the machine.

4.1 Human-computer interaction

If there is one characteristic which is typical of computer-assisted explorative mathematics, it is the interaction with the computer. Of course, the history of mathematics is full of interactions between mathematicians and non-human instruments. The most frequently used is the pencil-and-paper method: one writes and develops a notation in order to develop some technique, in order to communicate a result, in order to make a computation etc. This writing act always involves a "coding" practice – the use of symbols, of drawings, of abbreviations etc – which is historically determined and depends on the reason(s) why one is using paper and pen. Also within computer-assisted explorative mathematics coding practices are involved. However, there is one major difference: whereas in the former case, the interpreter is always a human, the interpreter in the latter case is a non-human, viz., the computer. It is exactly this difference which affects the coding practice and hence also the interaction. Most importantly, one is in need of a complex of interfaces – programming interfaces and output interfaces – which are used as communication devices in between the computer and the human. This requires new specialized kinds of knowledge and skills which take into account the fact that one is communicating with a non-human. First of all, one needs (a) suitable "language(s)" for the communication. It is exactly for this reason that we have seen the development of specialized software packages like Mathematica or Maple. It is also exactly for this reason that extensive use is being made from visualization techniques that allow the mathematician to come to terms with the vast amount of data generated by the computer. Secondly, and perhaps more importantly, one needs a good "format" to talk with the machine, viz. one needs good algorithms that are adapted to what the machine is good at *and* what it is bad at. I already provided the example of the development of pseudo-random generators, a type of algorithms that, except for some very rare instances, was never really considered before in the history of mathematics. Lehmer talks in this context about the need for an "idiot" approach: one needs algorithms that are executable by an idiot who cannot rely, for instance, on educated guesses and hence needs to be given instructions for every possible "behavior". In this context, one can, for instance, think of the SRT algorithm for division: it replaces the educated guessing we rely on for long division by a numerical table. Hartree, another computer pioneer, understood this new way of developing algorithms for the idiot as a real challenge for the future:

> [I]n programming a problem for the machine, it is necessary to take a "machine's-eye view" of the operating instructions, that is to look at them from the point of view of the machine which can only follow them literally, without introducing anything not expressed explicitly by them, and try to foresee all the unexpected things that might oc-

> cur in the course of the calculation, and to provide the machine with the means of identifying each one and with appropriate operating instructions in each case. And this is not so easy as it sounds; it is quite difficult to put oneself in the position of doing without *any* of the hints which intelligence and experience would suggest to a human computer in such situations

Today, part of these problems are already dealt with through developments in programming. For instance, the compiler allows to detect a whole range of syntactical and semantical errors by means of parsers, the symbol table and the semantic analyzer. Viz., part of these problems have already found a formal solution. However, some of them haven't and require the development of special more heuristic and statistical techniques. For instance, on the hardware level, it are the laws of statistics that govern part of how the memory is cached.

This requirement to look at a problem also from the machine's eye has resulted in important progress both within computer science and mathematics. In computer science, for instance, the development of compiler techniques allow to help the programmer to specify his algorithm in the machine's language. Within mathematics, an obvious example are the advances being made within numerical analysis since the rise of electronic computing: it fills the need for good and efficient iterative approximation techniques that "work" for the machine. For instance, in the 50s, when electronic memory was still very limited, new iterative algorithms were being used that allow to reduce the size of the memory needed during computation (see e.g. [9]).

4.2 Internalization

In the early years of digital general-purpose computing there were two major bottlenecks. The first was the programming bottleneck: even though computation speed was increased with several others of magnitude, the lack of programming languages and compilers meant that the setting up of a problem could take several days if not weeks. The second bottleneck was the memory bottleneck: electronic memory was very limited so one had to rely mostly on external memory devices like the punched cards. If additional memory was required during computation, this reliance on mechanical memory seriously slowed down the computation. Viz., the speed of memory storage and retrieval wasn't adapted at all to that of the computation. [11] As these two bottlenecks got steadily and partially resolved, it became possible to store internally into the machine more and more subroutines and the process of the so-called "algorithmization" of knowledge could be started. Also within mathematics one can observe a steady "algorithmization" of mathematical knowledge and skills. Some famous examples are the Zeilber-Gosper algorithm and the PSQL algorithm. The result of this is that today we have huge libraries of mathematical subroutines at our availability where algorithms like the Zeilberger-Gosper algorithm, can simply be called by their name.

[11] Today the speed of read and write operations is still a major topic in hardware design.

This steady process of internalization evidently goes hand-in-hand with human-computer interaction: since more and more processes are assigned to the machine rather than to the human, the boundaries between what is done by the machine and what is done by the human are more and more blurred. For instance, in the early years of digital general-purpose computing the machine was concerned only with brute-force mathematical calculations and the human did most of the exploration. Nowadays, part of this exploration is internalized into the machine. Indeed, going back to our simple example of the ENIAC computations of the digits of π and e, it is clear that today, no human would bother to do the statistics on the digits. The machine will generate both the digits as well as analyze and study them before something is returned to the human.

4.3 Time and processes

One final characteristic that I would like to consider briefly here, is the significance of time and processes within computer-assisted mathematics.[12] In a study which attempts at differentiating between so-called "computer science thinking" and "classical" mathematical thinking, Knuth sampled nine mathematical text books in order to get a firmer grip on the possible differences. His conclusions are [26]:

> Computer scientists will notice [...]that two types of thinking are absent from the examples we have studied [...] In the first place, there is almost no notion of "complexity" or economy of operation in what we have discussed. Bishop's mathematics is constructive, but it does not have all the ingredients of an algorithm because it ignores the "cost" of the constructions. [...] The other missing concept is related to the "assignment operation" :=, which changes values of quantities. More precisely, I would say the missing concept is the dynamic notion of the *state* of a process: "How did I get here? What is true now? What should happen next if I'm going to get to the end?" Changing states of affairs, or snapshots of a computation, seem to be intimately related to algorithms and algorithmic thinking. Many of the concepts of data structures [...] depend very heavily on an ability to reason about the notion of process states, and we rely on this notion also when studying the interaction of processes that are acting simultaneously.

Knuth's conclusion is in a certain sense not surprising: whereas "classical" mathematics seems to have no notion of process or time, within computer science everything is about time and processes! Indeed, a computation is executed as a dynamical process that develops and changes over time. One of the reasons why time is really an issue within computing is the fact that all computations on a standard computer are finite both with respect to time and space. This

[12] I am in fact very much indebted to Maurice for having drawn my attention to this fundamental difference between mathematics and computer science in a report he wrote for my dissertation.

stands in sharp contrast with mathematics where the infinite rules. In a report describing the state of computer science and engineering research this difference is described as follows [1, p. 9]:

> Mathematics deals with theorems, infinite processes, and static relationships, while computer science emphasizes algorithms, finitary constructions, and dynamic relationships. If accepted, the frequently quoted mathematical aphorism, the system is finite, therefore trivial, dismisses much of computer science.

This time dimension of computations becomes more explicit in the light of the fact that most computational processes are often not reversible and this in sharp contrast to mathematics. As Margenstern explains [33, p. 645]:

> Mathematical theories make use of reversible time; how can they exhaust nature, which is not at all reversible? 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.

The introduction of the arrow of time into computations combined with the finiteness of the machine also affects the mathematics that is done with such computations. For instance, if one is working with iterations over the reals, the fact that one has to work with finite approximations of these reals, combined with the fact that one is squeezing thousands or even millions of computations in a short time span, necessitates a study of the error propagations that may occur along the process. In fact, it is this problem that lies at the very foundation of the study of, for instance, non-linear equations like the quadratic iterator that I mentioned before: it was studied by Ulam and von Neumann (See e.g. [**?**, pp.3–4]). The processual and dynamical character of computation is reflected both on the hardware as well as on the programming level. We already saw the example of the assignment statement on the programming level in Knuth's quote. That this processual character of computations needs to be reflected also on the programming level, was already understood by von Neumann. Indeed, in a series of reports which introduces the well-known flowchart notation, he explains:

> [C]oding is not a static process of translation, but rather the technique of providing a dynamic background to control the automatic evolution of a meaning"

In fact, as he explains elsewhere, it is exactly this dynamical nature of computation that results in the need for logical control:

> [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.

On the hardware level, standard CPU design relies on a central clock signal that is emitted to all the hardware units. Without that central pulse which is used to control time, our current architectures would not be able to sequence and synchronize their operations let alone be able to "remember something".

5 Some (new and open) problems

Faced with a new situation created by the introduction of the computer, it becomes necessary for the mathematician who wants to use a computer in his research to reflect on a wide range of (new) problems or challenges that, at least, partially integrate a way of thinking that is more akin to computer science because it is rooted in characteristics which are typical of "using a computer". I already indicated some important developments within mathematics that are directly rooted in this new situation, like for instance within the field of numerical analysis or the study of randomness. In what follows I will sketch some (new and open) problems that (can) arise within a context of computer-assisted mathematics in more detail.

5.1 Mathematical understanding

It is well-known that results coming from computer-assisted mathematics, especially so-called computer-assisted proofs, are not very much appreciated by a part of the mathematical community. One important reason for this is that it is claimed that such results do not provide any understanding of the problem resolved with them. One famous example in this context is the computer-assisted proof of the four color theorem. In this context, Bonsall, a mathematician at the university of Edinburgh stated [**?**, p. 14]:

> It is no better to accept without verification the word of a computer than the word of another mathematician [...] We cannot possibly achieve what I regard as the essential element of proof – our own personal understanding – if part of the argument is hidden away in a box. [...] Perhaps we are seeing the birth of a new kind of computer-assisted quasi-mathematics, but is has no place in the science of mathematics and if it is to survive must develop its own scientific ethos – perhaps more akin to the experimental sciences.

Another mathematician, Ian Stewart, complains not only about the fact that part of such proofs are hidden but also about their lack of structure, viz. (Quoted from [29, p. 41]):

> [Such proofs do] not give a satisfactory explanation *why* the theorem is true. This is [...] mostly because it is so apparently structureless. The answer appears as a kind of monstrous coincidence. Why is there an unavoidable set of reducible configurations [within the four-color problem]? The best answer at the present time is: there just is. The proof: here it is, see for yourself. The mathematician's search for hidden structure, his pattern-binding [sic] urge, is frustrated"

The fact that with computer-assisted proofs, part of the proof is generated by a machine and mediated by a very lengthy code implies that, first of all, such computer-assisted proofs are far from being "elegant" and very lengthy and, secondly, that part of the proof is in fact not "surveyable" by a human. Moreover, such proofs are mostly based on a case-by-case analysis and, as such, are in need of a more non-uniform approach which is usual business in computer science but not in mathematics. For instance, for the four-color theorem over 1500 cases were derived! Add to this that such proofs are considered to be more error-prone because of their length and the programming and hardware involved and the mathematician is left with a feeling that this cannot be a "good" insightful proof. It is certainly true that the understanding one gains from, say, one of the classical proofs of the Pythagorean theorem cannot be identical to the understanding one gains from a proof that contains hundreds of pages of programs and millions of computations. And it is indeed not very insightful to "have a look" at the over 1500 cases of the original proof of the four color theorem just as there is no insight into why there are x cases rather that $x - 1$ cases!

Does this mean that there is no understanding whatsoever to be found in proofs such as that of the four color theorem? I do not think so. Rather it is my view that they provide a different kind of understanding. Indeed, from the perspective of a computer-assisted proof, it makes no sense to ask why a given problem reduces to x cases and not to y but it does make sense to follow the overall structure of such proofs as sketched in their published versions and to see how and why this semi-algorithmic structure works. To put it in the words of Borwein, a well-known advocate of experimental mathematics [3]: "*[T]he act of programming – if well performed – always leads to more insight about the structure of the problem.*"

This problem of what kind of understanding computer-assisted proofs such as the four-color theorem convey is a non-trivial problem which will have to be addressed properly by the mathematical community especially if, in the the future, more and more mathematical results would be proven in this way.

5.2 Hidden algorithms and explorations

One important characteristic of computer-assisted mathematics is that part of the mathematics required in tackling a given problem is internalized into the

machine. I discuss two of them in more detail here.[13] First of all, with current programming environments, much of the algorithms being used are no longer explicitly programmed by the mathematician who uses them but are instead called by their name. This means that part of the knowledge that is used within computer-assisted mathematics is unknown to the mathematician unless he/she has bothered him/herself with looking at the details of the subroutines called by the main program. This is even more problematical in the light of software that is not open source, like Mathematica. As Joris van der Hoeven, mathematician and developer of GNU TeXmacs and Mathemagix explains:[14]

> As a mathematician, I am deeply convinced that only free programs are acceptable from a scientific point of view. I see two main reasons for this:
>
> - A result computed by a "mathematical" system, whose source code is not public, can not be accepted as part of a mathematical proof.
> - Just as a mathematician should be able to build theorems on top of other theorems, it should be possible to freely modify and release algorithms of mathematical software.
>
> However, it is strange, and a shame, that the main mathematical programs which are currently being used are proprietary. The main reason for this is that mathematicians often do not consider programming as a full scientific activity. Consequently, the development of useful software is delegated to "engineers" and the resulting programs are used as black boxes.

In a certain sense, the use of algorithms without having gone through all of their details is comparable to referring to common or accepted mathematical knowledge without going through every of its details. However, if this knowledge is simply not free to access because, then I think that the challenges posed by computer-assisted proofs are rather small as compared to those posed by mathematical software that is not open-source! If in the future, computer-assisted mathematics becomes more important, this problem of hidden knowledge in combination with that of software licenses and patents will become a very important one, touching upon major issues such as scientific reproducibility and patentability.

A second problem that I would like to discuss briefly here concerns the idea of letting the computer do part of the exploration. It is a well-known practice that, when one uses a computer today, one will probably instruct it not to provide the mathematician with all data computed but with a certain selection of them, be it by way of visualizations, plots, statistics, etc. One important

[13] Note that the above problem of mathematical understanding is also partially rooted in the internalization of knowledge inside the machine.

[14] Extracted from `http://www.texmacs.org/tmweb/manual/webman-about.en.html` on February 24, 2014.

problem here is that the more of the exploration is delegated to the machine, the more assumptions one might put into the programming hence running the risk of missing out on some essential information. One example in this context comes from Wolfram who was studying so-called mobile automata, a class of computational devices which differ from cellular automata in that they do not update all cells in parallel but one at a time. Instead of looking explicitly at the "raw" behavior of these automata, Wolfram automated his search for a particular type of behavior [40]:

> [B]eing convinced that more complicated behavior must be possible, [...] I wrote a program that would automatically search through large numbers of mobile automata. I set up various criteria of the search, based on how I expected mobile automata could behave. And quite soon, I had made the program search a million mobile automata, then ten million. But still I found nothing. So then I went back and started looking by eye at mobile automata with large numbers of randomly chosen rules. And after some time what I relayed was that with the compression scheme I was using there could be mobile automata that would be discarded according to my search criteria, but which nevertheless had interesting behavior.

As becomes clear from this example, by internalizing part of the exploration one is always making certain assumptions which can result in errors. Another example is the use of Monte Carlo methods: since the 40s this has become a standard method in computer-assisted science in general. However, contrary to the early years of the Monte Carlo method, one only rarely questions the statistical assumptions underpinning this method.

The internalization of knowledge inside of the machine implies a number of problems that are far from trivial. The bottom line with these type of problems seems to be this: the more knowledge and assumptions that are hidden inside the machine, the more important it becomes for the mathematician to be critical about the knowledge he/she is presupposing. I consider this problem as a real treat to computer-assisted mathematics, at least as long as there is no critical awareness of the problem amongst mathematicians.

5.3 Finiteness, time and unpredictability

Perhaps one of the most important differences between mathematics and computer science is the fact that computer science deals with computational processes that develop over and are limited by time and space. Hence, it should not be surprising that this feature results in several problems for computer-assisted mathematics.

One important side-effect of the time-sensitive character of computations is their unpredictability. As Hamming explains [21]:

> 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.

This unpredictability is not just some theoretical problem or property. Indeed, it is in fact this unpredictability that usually brings mathematicians to the computer: it is because one cannot predict the outcome of a certain computational problem that one needs to rely on and trust the machine's abilities. However, many such problems, when programmed, may contain infinite loops or need an unreasonable if not infinite amount of time and/or space and such that this cannot be predicted by the mathematician. This necessitates the need for developing local programming strategies that are often "experimental" in nature, viz., they do not necessarily result in a (correct) solution and often require adjustments in the light of new computational results. For instance, Hales had to experimentally determine several constants in the process of his proof of the sphere packing problem [7]:

> Hales remarks for instance that "The constant 2.51 was determined experimentally to have a number of desirable properties", and similar experimental determinations recur repeatedly throughout the paper. Several of the initial decisions had to be modified in the light of later calculations.

Another problem that comes up in this context can be illustrated with the following example coming from my own computer-assisted research on a class of computational devices known as tag systems. Also here it was the unpredictability of the computational processes of these devices that made it necessary for me to include certain limitations into the programs used to study these devices and which instructed the computer when it should give up on a certain tag system. For instance, in studying the behavior of tag systems with arbitrary initial words of length 300, I included the limitation that the program should stop if (1) after 10,000,000 computational steps the tag system had not halted or become periodic and (2) one of the produced words became longer than 15,000,000. Initial words that resulted in more than 10,000,000 computational steps were tentatively classified as possible immortals. But what kind of derivations can one make on the basis of information that is based only on a finite piece of information of a (possibly) infinite dataset? In how far are, for instance, the results from a statistical analysis based only on the first part of a computation representative for the whole computation?

These kind of problems are not only characteristic for computer-based research but for explorative mathematics in general: experimental research on the Riemann-zeta function, the twin prime conjectures, the Goldbach conjecture or the Collatz problem all suffer from the problem that one has information only

about a finite subdomain of the (possibly) infinite domain. Consider for instance the Collatz function C. Given an integer n_0, if n_0 is odd compute $n_1 = 3n_0 + 1$ if not, compute $n_1 = n_0/2$. If n_1 is odd, compute $n_2 = 3n_1 + 1$, else, compute $n_2 = n_1/2$, etc. The Collatz conjecture states that for any n_i, there is an m_j such that after m_j iterates of C on n_i it will end in the loop 1, 2, 4. Now, assume we want to compute C for some very large integer. We start the computation on our computer but after weeks of iterations we have an overflow. What would this tell us? Conversely, what can we derive from the fact that the conjecture has been verified for 5×2^{60} integers? It is in such experimental context that one is in need of a "conduct of good behavior", a certain standard of when one has reasons to make a conjecture and when not and one should always be extremely careful in making generalizations based on a finite number of instances.

Another problem, which is directly related to the finiteness of the computer, comes up in the context of computations over the reals and is related to the Mandelbrotset. Given the function:

$$c \to c^2 + c, \ c \in \mathbb{C}$$

the Mandelbrot set is defined as:

$$M = \{c \in \mathbb{C} | c \to c^2 \to c^2 + c \to \ldots \text{remains bounded}\}$$

The set is most well-known for its intricate boundary which gives rise to very beautiful visualizations. One important question, that was formulated by Roger Penrose, is whether it is decidable for any c whether or not it belongs to the Mandelbrotset. There are now two major competing models of computation over the reals to address this and other related problem:

> [The bit model] reflects the fact that computers can store only finite approximations to real numbers. Roughly speaking, a real function f is computable in the bit model if there is an algorithm which, given a good rational approximation to x, finds a good rational approximation to $f(x)$. The second approach is the algebraic approach, which abstracts away the messiness of finite approximations and assumes that real numbers can be represented exactly and each arithmetic operation can be performed exactly in one step. The complexity of a computation is usually taken to be the number of arithmetic operations (for example, additions and multiplications) performed. The algebraic approach applies naturally to arbitrary rings and fields, although for modeling scientific computation the underlying structure is usually $\mathbb{R}$ or $\mathbb{C}$.

The second approach is most known through the work of Blum, Shub and Smale: within their model, the Mandelbrotset turns out to be undecidable. However, the drawback of the model is that functions which one would consider intuitively as being decidable, like, for instance, a simple transcendental function such as e^x, also turn out to be undecidable under this model. This is not the case with the bit model. In this latter model, the undecidability of the Mandelbrot set

is still an open problem. One interesting consequence of this model is that *if* a function f is uncomputable than there is no good rational approximation of f. As such, the undecidability of the Mandelbrot would imply that we cannot rely on our computer-generated visualizations of the set. Indeed, since any such visualization is but a finite approximation of the "real" set, we cannot rely on it and hence also not on the numerous conjectures that surround the set and which are based on these visualizations.

5.4 Unreliability

From the previous sections it becomes clear that some of the characteristics of computing on a machine result in a number of problems within computer-assisted mathematics that need to be taken into account in some or the other way by the mathematician who uses the computer in his/her research. All of these different problems contribute to a feeling that the results from computer-based research are quite unreliable. The fact that usually hundreds or thousands of lines of code are involved, the fact that the human him/herself can often only wait and see, the fact that one does not have access to all data generated during the process, the fact that one may encounter truncation errors, etc indeed do not give an impression of high reliability. Add to this that many results are "experimental" results and it becomes understandable that many a mathematician turns his/her back to the results that are machine-aided. On the other hand, however, one cannot deny that the computer in fact gives us the possibility, to state it again in Lehmer's words, "*to explore the terrain that has been staked out so freely and that something worth proving will be discovered in the rapidly expanding universe of mathematics*". The choice is there to make for the mathematician but if the choice is in favor of the machine it is clear that one needs to deal in some or the other way with this problem of unreliability that becomes so explicit in this context. Two extreme positions have been proposed in this context and relate back to the tensions I recounted in Sec**??** those between formalism vs. experimentalism; rigorous vs. non-rigorous math, etc as highlighted in the contrast between Hilbert's work and that of the late von Neumann.

One extreme position is represented by Doron Zeilberger, a well-known mathematician and strong supporter of computer-assisted mathematics. He has claimed that, even though today [41]:

> the mathematical faith is *thou shalt prove everything rigorously* [...] a new testament is going to be written. Although there will always be a small group of "rigorous" old-style mathematicians [...] who will insist that the true religion is theirs, and that the computer is a false Messiah, they may be viewed by future mainstream mathematicians as a fringe sect of harmless eccentrics [...] In the future, not all mathematicians will care about absolute certainty, since there will be so many exciting new facts to discover. We will have (both human and machine) professional *theoretical* mathematicians, who will develop conceptual paradigms to make sense out of the empirical data, and

> who will reap Fields medals along with (human and machine) *experimental* mathematicians. [...] As absolute truth becomes more and more expensive, we would sooner or later come to grips with the fact that few non-trivial results could be known with old-fashioned certainty. Most likely we will wind up abandoning the task of keeping track of price altogether, and complete the metamorphosis to non-rigorous mathematics.

This is quite a strong claim: Zeilberger is convinced that human "proofs" as certificates of truth will be abolished and replaced by a non-rigorous mathematics because few non-trivial truths that can be proven by short and human proofs will be left. Again, it is impossible at this point to evaluate this prediction, but the idea that mathematics would *completely* abandon the very idea of proof as a guarantee of certainty, is precarious. This, however, does not mean that there will be no situations where mathematical results, which are true only to a very high degree of certainty, will be more easily accepted. It is already a common practice to use for instance probabilistic algorithms and, in the context of computer-assisted proofs, the idea of corroboration, where different groups of researchers prove the same result independently from each other, has also already been suggested. For instance, MacPherson, who was an editor of the *Annals of Mathematics* at the time that Hales' proof was being reviewed wrote to Hales about Fejes Thoth, the head of the team of 12 human reviewers of the proof [20]:

> Fejes Toth thinks that this situation will occur more and more often in mathematics. He says it is similar to the situation in experimental science - other scientists acting as referees can't certify the correctness of an experiment, they can only subject the paper to consistency checks. He thinks that the mathematical community will have to get used to this state of affairs.

An opposite response to the challenges posed by the computer for mathematics, is to take the side of full rigorous mathematics by means of formalized proofs with the help of interactive proof assistants such us HOL or Isabel. The four-color theorem, for instance, has been formalized in this way by Gonthier [18] and also Hales and others are now working for some years on the so-called Fly-Speck project which aims at formalizing the computer-assisted proof of Kepler's conjecture [19]. The reason why people like Gonthier and Hales are working on such proofs is rooted in the uncertainty associated with lengthy computer-assisted proofs. By means of formalization, it is believed that possible errors are excluded since every single logical step is (supposedly) verified by the proof assistant [19]:

> A formal proof is a proof in which every logical inference has been checked, all the way back to the foundational axioms of mathematics. No step is skipped no matter how obvious it may be to a mathematician. A formal proof may be less intuitive, and yet is less susceptible

> to logical errors. Because of the large number of inferences involved, a computer is used to check the steps of a formal proof.

This approach of formalized proofs however is not only applied to lengthy computer-assisted proofs. The developments of automated verification and proving has resulted in initiatives like the Mizar project which aim at formalizing the whole of mathematics. The motivations are the same: a believe that, partially due to the introduction of the computer into mathematics, mathematical knowledge is becoming too complex and, as a consequence, that there might be a tendency towards non-rigouresness. In the QED manifesto, which proposes the formalization of all mathematics, this is stated as follows [37]:

> [T]he increase of mathematical knowledge during the last two hundred years has made the knowledge, let alone understanding, of all of even the most important mathematical results something beyond the capacity of any human. For example, few mathematicians, if any, will ever understand the entirety of the recently settled structure of simple finite groups or the proof of the four color theorem. Remarkably, however, the creation of mathematical logic and the advance of computing technology have also provided the means for building a computing system that represents all important mathematical knowledge in an entirely rigorous and mechanically usable fashion. The QED system we imagine will provide a means by which mathematicians and scientists can scan the entirety of mathematical knowledge for relevant results and, using tools of the QED system, build upon such results with reliability and confidence but without the need for minute comprehension of the details or even the ultimate foundations of the parts of the system upon which they build.[...]

Also here it seems precarious to believe that, in the future, mathematical knowledge will become completely formalized in some computer-based system. Besides the fact that there is a problem here of infinite regress, indeed, "*who will control the controller*",[15], "*[i]t is a large labor-intensive undertaking to transform a traditional proof into a formal proof*" [19]. Hence, just as one can wonder whether your average mathematician will really bother about non-rigorous mathematics à la Zeilberger, one can wonder whether he/she will be bothered about formalizing their results.

These two opposite alternatives, formalized and non-rigorous mathematics, are two sides of the same medal. They are both rooted in a believe that mathematics is becoming less reliable and rigorous, partially due to the introduction of the computer. At this point, it is not really clear at all if any of these two alternatives will really become a dominant practice of future mathematics in one or the other from. But is it really desirable that we evolve to any of these two alternatives? Faced with the new situation in mathematics, I believe that mathematics can only gain if both sides, rather than one dominating the other,

[15] Remark from Maurice Margenstern during MCU 2013

as was often the case in the past, would try and complement/advance each others findings.

6 Some afterthoughts

Computer science has many different historical roots and mathematics is certainly one of them: formalization and algorithms were and are a part of the mathematical discourse and this long before the computer inspired the formation of a new discipline known as computer science. Today we see that a converse influence is manifesting itself: the computer is also starting to affect mathematics. Computer-assisted exploration is resulting in a revival of so-called experimental mathematics, a practice which is certainly not new to the history of mathematics but has been pushed to the background for at least the last two centuries. It is within such practice that algorithms and non-uniform methods rather than general theories are at play.

It is thus not surprising that explorative computer-assisted mathematics has more in common with a computer science way of thinking than "classical" mathematics. The reason for this is not the mere fact that extensive computation is now possible, but is also rooted in the use of a physical, finite machine. From a contemporary perspective, the explicit focus in this paper on how the machine itself affects mathematics may seem a bit old-fashioned. Indeed, today, there is a tendency towards abstraction up to the development of interfaces which are made as "user-friendly" as possible, hiding at the same time that one is using a technological device. In fact, it has become rather common to see computation as something that transcends the computer itself and can be found anywhere (See e.g. [12]). Evidently, we are very much in need of such layers of abstraction. Amongst others, they allow us to overcome the programming bottleneck that characterized the early machines. However, if the addition of such layers goes hand-in-hand with forgetfulness about the constraints imposed by the machine or, more generally, the physicality of computation, on our (mathematical) thinking, one also gives away control, not only to the machine, but also, more importantly, to the developers of these different layers of abstraction

The usage of the computer implies that one needs to develop algorithms that *really* work, also, from the machine's eye and for which, as a consequence, mathematical elegance is constrained by the need for procedures that have to be scientific before becoming an art.[16] It also means that a new range of problems need to be faced by the mathematical community, problems that originate in, amongst others, the time-based character of computations which has become

[16] I am referring here to Knuth's Turing award lecture titled "Computer programming as an art" in which he reflects on the multiple ways in which "science" and "art" can be used in the context of programming, and beyond. During that lecture he stated: "*Science is knowledge which we understand so well that we can teach it to a computer; and if we don't fully understand something, it is an art to deal with it. Since the notion of an algorithm or a computer program provides us with an extremely useful test for the depth of our knowledge about any given subject, the process of going from an art to a science means that we learn how to automate something.*" [25]

more explicit through their execution on a very fast but limited machine; the internalization of procedures inside of the machine and the fact that part of the work is, literally, out of control of the human. These problems result in two extreme positions with respect to computer-assisted explorative mathematics and the more general observation that mathematical knowledge is becoming too complex to guarantee rigor: on one side, there are those who argue for the complete formalization of mathematical knowledge and thus "automated rigor", on the other side, there are those who want the exact opposite, viz. "automated non-rigor". Interestingly, as I argued in Sec. **??**, one can trace a similar opposition within the short history of computer science itself, which, very roughly speaking, boils down to the fact that computer science is as much about engineering as it is about mathematics.[17] The formal verification debates, with its high point in the 80s, is perhaps one of the more explicit and harsh exemplifications of this opposition. However, just as Lehmer's two styles need not be exclusive in mathematics, the same holds true for computer science. In fact, within computer science, both engineering and formalization are so intricately tied together through the physicality of computation that one can only feel regret when these two "styles" are regarded as each others opposite. One can only hope that as computer science matures, that the disciplinary fences that have already been built, will not result in a complete separation of both styles. This can only result in an impoverishment of the field.

References

[1] Arden, B.W. (ed.), *What can be automated?: The computer science and engineering study*, MIT Press, 1980.

[2] Bonsall, F.F., *A down-to-earth view of mathematics*, in: The American mathematical monthly, vol. 89, nr. 1, 1982, pp. 8–15.

[3] Borwein, J., *Implications of experimental mathematics for the philosophy of mathematics*, in: Gold, B. and Simons, R.A., Proof and other dilemmas: Mathematics and philosophy, Mathematical Association of America, 2008, pp. 33–60.

[4] Brady, A.H., *The determination of the value of Radó's noncomputable function σ for four-state Turing machines*, Mathematics of Computation **40** (1983), no. 162, 647–665.

[5] Braverman, M. and Cook, S., *Computing over the reals: Foundations for scientific computing*, in: Notices of the AMS, vol. 53, nr. 3, 2006, pp. 318–329.

[17] In fact, it has been argued that there are in fact three styles within computer science: a mathematical style characterized by theoretical work, an engineering style characterized by design and a scientific style characterized by abstraction and modeling [11].

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

[7] Conway, J.H.; Goodman-Strauss, C.; Sloane, N.J.A., *Recent progress in Sphere Packing*, Contemporary Mathematics, to appear, available at: comp.uark.edu/ strauss/papers/sphere.pdf?.

[8] Church, Alonzo, *A set of postulates for the foundation of logic (second paper)*, Annals of mathematics, vol. 34, nr:4, pp. 839–864, 1933.

[9] Clenshaw, C., *Polynomial approximations to elementary functions* in: Mathematical Tabels and Other Aids to Computation, 8, nr. 47, 1954, pp.143–147

[10] Corry, L., *Number crunching vs. number theory: computers and FLT, from Kummer to SWAC (1850–1960) and beyond*, Archive for the history of Exact Sciences, 62, 2008, pp. 393–455.

[11] , Denning, P.J.; Comer et al., *Computing as a discipline*, Communications of the ACM, vol. 32, nr. 1, 1989, 9–23.

[12] , Denning, P.J., *Computing as a natural science*, Communications of the ACM, vol. 50, nr.7, 2007, pp. 13–18.

[13] De Mol, L., *The proof is in the process. A preamble for a philosophy of computer-assisted mathematics*, M.-C., Galavotti et al (ads.), New directions in the philosophy of science, Springer, in print.

[14] Dijkstra, E.W., "On the design of Machine Independent Programming Languages, Report MR34, Stichting Mathematisch Centrum, Amsterdam, 1961.

[15] Dijkstra, E.W.,*Porgramming as a discipline of mathematical nature*, The American Mathematical Monthly, 81:6, 1974, pp. 608–612.

[16] Dijkstra, E.W., *How computing science created a new mathematical style*, EWD1073, 1990.

[17] Gandy, R., *The confluence of ideas in 1936*, in: Herken, R. (ed.), The Universal Turing machine, Oxford University Press, Oxford, 1988, pp. 55–111.

[18] Gonthier, G., *Formal proof – The four color theorem*, Notices of the AMS, vol. 55, nr. 11, 2008, pp. 1382–1393.

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

[20] Hales, Th., *The Flyspeck project*, http://code.google.com/p/flyspeck/wiki/FlyspeckFactSheet

[21] Hamming, R.W. (1965), "Impact of Computers", *The American Mathematical Monthly*, 72: 1–7.

[22] Hamming, R., *One man's view of computer science*, Journal of the ACM, vol. 16, 1969, pp. 3–12.

[23] Hilbert, D., *Naturerkennen und Logik*, Naturwissen, 18, 1930, pp. 959–963.

[24] Knuth, D., *Computer science and its relation to mathematics*, American Mathematical Monthly, 81:4, pp. 323–343.

[25] Knuth, D., *Computer programming as an art*, Communications of the ACM, vol. 17, nr. 12, 1974, pp. 667–673.

[26] Knth, D., *Algorithmic thinking and mathematical thinking*, The American Mathematical Monthly, vol. 92, nr. 3, 1985, pp. 170–181.

[27] Knuth, D., *Algorithmic themes*, in: Duren, P. et al (eds.), A century of mathematics in America, Part I, American Mathematical Society, 1988, pp. 439–445.

[28] Krantz, S., *Letter to the editor*, The American Mathematical Monthly, vol. 91, nr. 9, 1984, 596–600.

[29] MacKenzie, D., *Slaying the Kraken: The sociohistory of a mathematical proof*, in: Social Studies of Science, vol. 29, nr. 1, 1999, pp. 7–60.

[30] Lehmer, D.H., *Mathematical methods in large scale computing units*, Proceedings of Second Symposium on Large-Scale Digital Calculating Machinery, 1949 (Cambridge, Massachussetts), Harvard University Press, 1951, pp. 141–146.

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

[32] Mancosu, P., Zach, R. and Badesa, C., *The Development of Mathematical Logic from Russell to Tarski, 1900 - 1935*, in: Haaparanta, L. (ed.), The Development of Modern Logic, Oxford University Press, New York, 2009, pp. 318 - 470.

[33] Margenstern, M. (2012), "Comment", in: H. Zenil, ed., *A computable Universe*, Singapore: Worldscientific Press, pp. 645–646.

[34] Naur, P., *The place of programming in a world of problems, tools, and people* in: Proceedings IFIP Congress 65, 1965, pp. 195–199. Also published in Naur, P., *Computing: a human activity*, ACM Press, 1992.

[35] Post, E., *Absolutely unsolvable problems and relatively undecidable propositions - Account of an anticipation*, 1965, in: Martin Davis, *The undecidable. Basic papers on undecidable propositions, unsolvable problems and computable functions*, Raven Press, New York, 1965, Corrected republication (2004), Dover publications, New York, pp. 340–433.

[36] von Neumann, J., *The Mathematician*, in: Heywood, R.B. (ed.), The works of the mind, University of Chicago Press, 1947, 180–196.

[37] *The QED Manifesto*, available at: `http://www.cs.ru.nl/ freek/qed/qed.html`

[38] Ulam, S., *John von Neumann 1903–1957*, Bulletin of the American Mathematical Society, vol. 64, 1958, pp. 1–49.

[39] Wegner, P., *Research paradigms in computer science*, Proceeding ICSE '76 Proceedings of the 2nd international conference on Software engineering, 1976, pp. 322–330.

[40] Wolfram, S., *A new kind of science*, Wolfram Media, Champaign, 2002.

[41] Zeilberger, D., *Theorem for a price. Tomorrow's semi-rigorous mathematical culture*, The Mathematical Intelligencer, vol. 16, nr. 4, 1994, pp. 11–18.