Seventy Years of Computer Science

Martin Davis(✉)

Courant Institute, New York University, New York, USA
martin@eipye.com

Abstract. A quick tour through my long career with emphasis on how computer science has affected me and how I have affected computer science.

Keywords: Turing · Copeland · Neural net · Recursive functions · Computability · ORDVAC · Presburger arithmetic · ACE · Test data adequacy · **NP** · Hypercomputation · Universal · SAT · Linked conjunct · Obvious inferences

I delivered my typewritten doctoral dissertation to the Princeton University registrar in May 1950, just seventy years before May 2020. In this essay I will try to provide a very personal survey of my interactions with computer science, as participant and as observer, over this seventy year period.

The title of my dissertation was *On the Theory of Recursive Unsolvability.* In 1935 Alonzo Church[1] had declared that the *recursive functions*, defined on the natural numbers, are precisely those that are algorithmically computable. It was Gödel who had defined this class, and Church's student Kleene had found various alternative formulations of the class. Church and Kleene had also studied a class of functions, the λ–*definable* functions, defined in a very different manner, and it had been proved that the two classes were the same. Meanwhile in England, Alan Turing's[2] formulation, in terms of what came to be called Turing machines, was proved by Turing to be equivalent to these other two. E. L. Post[3] had a formulation very close to Turing's that he had developed independently. Post had also worked on a quite different formalism, his *canonical* and *normal* systems of productions, during the 1920s and proposed them as providing yet another formulation he expected to be equivalent to the other three.

In my dissertation I studied various aspects of the theory of recursive functions basing myself on Kleene's version of Gödel's formulation. I proved that Post's canonical systems were equivalent to the other formulations and, using Post's reduction of canonical to normal systems, obtained an unsolvable problem for normal systems. With this problem as a basis, I obtained my first unsolvable

[1] He was my adviser.

[2] Church was Turing's adviser as well. But Turing's computability paper was written before he became a Princeton student to work with Church.

[3] He was my teacher when I was an undergraduate at City College in New York.

A. Blass et al. (Eds.): Gurevich Festschrift, LNCS 12180, pp. 105–117, 2020.
https://doi.org/10.1007/978-3-030-48006-6_8

problem involving Diophantine equations. Was this computer science? A mile and a half away from the Princeton campus, von Neumann's computer at the Institute for Advanced Study was being built in 1950, not to be operational for another two years. But I was aware of this only dimly if at all. In any case, the thought that it had any relation to my dissertation would never have occurred to me. Still Gödel's advance to what Kleene called "general" recursion from the earlier "primitive" recursion corresponds to including *while* loops in programming languages in addition to simple looping constructs. A substantial part of my dissertation was to appear, couched in the language of Turing machines, in my book, *Computability and Unsolvability*, of 1958. In Computing Reviews this book was called "one of the few classics of computer science" when Dover reprinted it in 1982. Also, Chomsky's hierarchy of classes of formal languages was based explicitly on Post's production systems.

In the fall of 1950, amid considerable turmoil in my personal life, I found myself amid the corn fields of southern Illinois, where I had moved to take up a postdoc position at the University of Illinois at Champaign-Urbana. In the spring semester I was able to teach a graduate course on recursive functions. I liked the intuitive feeling of Turing machines and decided to base my presentation on them rather than on the general recursions in my dissertation. In showing that complicated algorithms could be coded for Turing machines, I was writing lots of specific Turing machine code on the blackboard. A freshly minted PhD in mathematics like me, Ed Moore, who had been auditing the course, came to the front of the room after one of the sessions and showed me how I could improve some of the code I had written. Then he said, "We have one of those across the street." He was referring to the ORDVAC, a computer built at the University of Illinois pretty much along the lines of von Neumann's machine in Princeton.

What brought me to the ORDVAC was the Korean War. When Truman decided to militarily oppose the invasion of South Korea from the north, a group of University of Illinois academics, mostly physicists, joined to try to use their scientific knowledge to aid the effort. I was recruited for this new organization, the Control Systems Laboratory, and I accepted the offer. To begin with, there was a heady brew of ideas in the air: Wiener's cybernetics, Shannon's theory of computation, and computers with their unknown potential. Eventually it was decided to build a prototype of a system in which the ORDVAC controlled physical devices. Specifically, it was to navigate 100 airplanes in real time. And the task of writing the program that would do this was given to me.

The ORDVAC was built around a William's memory in which data was stored as electric charge on the surface of cathode ray tubes (CRTs). There were 40 small CRTs each capable of storing 1024 bits in a 32×32 array. So the total memory was 5 KB. A memory access or an addition required 40 ms, and a multiplication or a division required a full millisecond. My program was to respond to "radar" information providing position information of the "planes" by computing a heading for each plane and transmitting it to the plane. The computation consisted of a sorting part, in which each radar input was matched with the corresponding plane, and a part in which the headings were computed.

I used a merge-sort algorithm for the first part. For the second part I had the help of a young physicist, Marius Cohen, who found an algorithm for computing the sine function to sufficient accuracy using just one division and no multiplications. All of this had to be interrupted regularly to intercept incoming radar data. I had to place these interruptions in the code based on a hand calculation using the published times required for the instructions to be carried out.

Programs for the ORDVAC were written in absolute binary machine language, with nothing like an assembly language available. The ORDVAC had no index register so control of loops required the code to operate on itself. Nevertheless I found programming lots of fun. Also it was clear to me that it was essentially the same activity that I had been engaged in in the classroom programming Turing machines. But it would be some time before I came to understand how intimate the connection is. It was fun, but it wasn't my own work. When I was able to obtain support for two years at the Institute for Advanced Study in Princeton, I eagerly seized the opportunity.

Back in Princeton with my pregnant wife Virginia, whom I had met and married in Champaign-Urbana, I was free to work out some of my ideas related to Gödel incompleteness. I also began working on the book that was to become *Computability and Unsolvability.* I wanted to show computability as a full-fledged subject in its own right with diverse applications. Basing it on Turing machines, I wanted the connections with computer practice to emerge. Before my two years at the Institute for Advanced Study were up, I was able to bring a complete handwritten manuscript to the typists. I knew that my handwriting had difficulties, but nevertheless I was dismayed by the poor job the typists had done. Every page needed many corrections, and when I brought the corrected typescript back to the typists, they refused to have anything to do with it. I suppose that after typing the work of such as Einstein, Gödel, and von Neumann, they felt entitled to ignore the needs of 26 year old mere visitor. The typescript languished in closets for over two years.

The terms of my support gave me the option of seeking summer employment, and we certainly needed the money. In the summer of 1953 I worked at Bell Labs, an easy commute from Princeton. My supervisor was Claude Shannon whose fundamental tract on information theory I had read in Champaign-Urbana. Ed Moore, who had first told me about the ORDVAC, was there as well. Shannon had designed a universal Turing machine with only two states. He raised the question: Can one provide a precise definition of universality? He pointed out that unless the input/output is carefully specified, a Turing machine might exhibit universal behavior only because of the way input data was coded. I liked the problem and wrote two papers about it.

For the following summer I had managed to receive funding for a project to program the decision procedure for Presburger arithmetic. I had permission to use the Institute for Advanced Study computer for the purpose. I completed the project, and we were off, driving across the country to Davis, California where I was going to be an assistant professor of mathematics at one of the campuses of the University of California. While there wasn't much intellectual activity in the

Davis mathematics department, in Berkeley, 80 miles away, the great logician Alfred Tarski led an outstanding group of young scholars. When I was invited to speak at the weekly colloquium of the Berkeley mathematics department, I took the opportunity to express my view of computability theory as an autonomous branch of mathematics. Tarski, who was in the audience, took strong exception during the discussion after my talk.

After Virginia and I were awakened by the onset of labor, our second son Nathan was born at home at 4AM. Other than us, the only person in the house was Nathan's two year old brother Harold. Virginia's obstetrician was far away in Berkeley, and we made do with an obstetric textbook. As I write, Harold and Nathan are men well into their sixties.

We remained in Davis for just one year and moved to Columbus, Ohio, where I was again an assistant professor of mathematics teaching elementary subjects. Again we left after a year. I eagerly accepted an offer from the Hartford Graduate Center that Rensselaer Polytechnic Institute had established in Eastern Connecticut. I remained there for three very fruitful years. With a faculty of perhaps a dozen, with only three of us teaching mathematics, it was quite an interesting place with mature students, quite different from those, not yet 20, that I had been teaching in Davis and Columbus.

The secretaries there were eager to be helpful and did an excellent job of turning my manuscript for *Computability and Unsolvability* into proper shape for being submitted to a publisher. McGraw-Hill offered me a contract and published it in their series on "Information Processing and Computers". In my preface I wrote:

> The existence of universal Turing machines . . . confirms the belief . . . that it is possible to construct a single "all-purpose" digital computer on which can be programmed . . . for any conceivable deterministic digital computer.

A quarter of a century was to go by before I came to understand just how intimate was the relationship between Turing's abstract model of a universal computer and physical computers, that the former was the progenitor of the latter.

The book was written from the point of view of computability as an independent discipline. This view was reinforced by a section on applications with chapters on logic, algebra, and number theory. Tarski's equally valid view, expressed when he objected to what I had said in my Berkeley talk, was that computability is a branch of definability theory which is part of mathematical logic.

Of course at the time this book was written, the academic discipline of computer science didn't yet exist. Nevertheless computer science is indebted to computability theory in several ways. It supplied two models of computation, the Turing macine and the register machine that proved useful in quantifying the asymptotic complexity of specific algorithms. Also, the complexity classes poly-time, NP-completeness, and the levels of the poly-time hierarchy were all defined by analogy with categories from computability theory.

In the summer of 1957 (a year before *Computability and Unsolvability* was published) there was a remarkable five week "Institute for Logic" at Cornell

University in Ithaca, 220 miles northwest of New York City. 85 logicians attended, almost all from the U.S. The influence of the newly developing world of computers was already evident. Alonzo Church lectured on "Application of Recursive Arithmetic to the Problem of Circuit Synthesis". I spoke about my program for Presburger arithmetic. Abraham Robinson discussed "Proving a Theorem (As Done by Man, Logician, and Computer)". Rabin and Scott's fundamental work on finite automata was presented. IBM sent a contingent of 13 to the "Institute". Among other things they presented FORTRAN, initiating controversy on whether the loss of efficiency in programming in such a "high level" language compared to using assembly language was supportable. Altogether of the 82 talks presented, 19 had a definite computer science aspect.

I had become friendly with Hilary Putnam, a young philosopher at Princeton, and we decided to share a small house in Ithaca with our families for the duration of the Logic Institute. We were together nearly every day, and this led directly to our fruitful collaboration. I had studied what I had called Diophantine sets. A set S of natural numbers is *Diophantine* if there is a polynomial $p(a, x_1, x_2, \ldots, x_n)$ such that $a \in S$ if and only if there are natural numbers $x_1, x_2, \ldots, x_n$ for which $p(a, x_1, x_2, \ldots, x_n) = 0$. In my dissertation I had conjectured that every set of natural numbers which is *listable*[4], in the sense that there is an algorithm that generates a list of the members of the set, is also Diophantine. I had taken a first small step towards proving this conjecture. It was easy to see that the truth of the conjecture would yield a solution to the tenth of the 23 problems that Hilbert famously had proposed in 1900. Hilary and I began working on this conjecture; we found a new approach yielding a nice theorem that we were pleased to present at the Institute. I would not claim that this was computer science, but it would be impossible to omit it from any account of our collaboration.

We were so pleased by what we had accomplished that we decided to seek funding enabling us to work together during the summer months. Because the experts regarded my conjecture as very unlikely to be true, we thought it would be hopeless to seek funding for that. So we decided instead to write a proposal for work on computer generated proofs of theorems. Specifically we proposed to work on a proof procedure for first order logic. Our proposed procedure was to include an algorithm for what has come to be called SAT, the satisfiability problem. We wrote a proposal, but it was too late for submission to the funding agencies if we hoped to work together the following summer. A friend of Hilary suggested we send it to the National Security Agency (NSA). Neither of us had heard of this agency, but, nothing to lose, we submitted it to them. A phone call came inviting me to visit the NSA's headquarters in Maryland. When I told them that I had never heard of the agency, they laughed and said their publicity office was doing a good job. The NSA was not to remain so obscure for long. It became front-page news when two of their people defected to the Soviet Union.

When we talked about our proposal, they made it clear that they had no interest in proof procedures for first order logic, but they were interested in SAT.

[4] Other terms for this notion are *recursively enumerable set* and *computably enumerable set.*

They warned me that it is a difficult problem and doubted that we could make much headway in one summer. However if we were willing to work only on SAT, they were prepared to fund our proposal. I agreed at once. Our report[5] submitted to them at the end of the summer, made no mention of the funding agency, as they had requested. The report introduced the technique of initially transforming a Boolean formula being tested for satisfiability into conjunctive normal form. This amounted to a list of disjunctive clauses, a format that came into widespread use. Various techniques were offered for satisfiability testing with examples.

The following summer, we had funding from an agency whose mission was to support fundamental science. Not constrained, we set to work on my conjecture that every listable set is Diophantine. We managed to find a proof of the weaker result that every listable set is exponential Diophantine, meaning that variable exponents were permitted in the algebraic expression.[6] For the part of our report related to my conjecture, see [18] pp. 411–430. While we were writing our report, we recalled that our proposal to the agency called for work on machine theorem proving. So using a selection of the algorithms from our report of the previous summer together with an exhaustive search of the Herbrand universe (a term I later introduced in [2]), we had our proof procedure. We wrote it up for our report, and, on a whim, I submitted it to the Journal of the ACM. They published it [14] and it is easily the most cited of my publications, the source of the "Davis-Putnam procedure". Hard copies of our reports to the two agencies are in an archive maintained by Donald Knuth. Julia told me that when she brought a copy of our report for the Russian mathematicians, they were astonished that the US Air Force funded research on Diophantine sets, research that was very unlikely to lead to any practical applications.

In the spring of 1959, I was surprised to receive a letter offering me a year appointment at the Institute for Mathematics and Mechanics at New York University (NYU). I had flirted with them before, but their offer at that time had been unsatisfactory. This institute was totally the creation of Richard Courant. A Jew, he had been expelled from the mathematical institute at Göttingen of

[5] The report is available at [18] as Appendix A pp. 374–408.

[6] Our proof had a flaw. It used the fact that there are arbitrarily long arithmetic progressions consisting entirely of prime numbers. This fact was only proved in 2004 (by Ben Green and Terrence Tao); so we had to call it a hypothesis. We wrote our work up for our funding agency, the Office of Scientific Research of the US Air Force. We also submitted it for publication to a mathematical research journal. In addition we sent a copy to Julia Robinson whose methods had greatly influenced our approach. To our delight she succeeded in modifying the proof so it did not need this as yet unproved proposition. We withdrew our paper, and the theorem was published with the three of us as authors. It followed from the new result that my conjecture would follow if a single polynomial could be found that satisfied two simple conditions that Julia had proposed. After the three of us had been trying for a decade to find such a polynomial, we learned that Yuri Matiyasevich, at the age of 22, had actually done it. His proof that his equation satisfied Julia's conditions, though quite elementary, was intricate and beautiful.

which he had been a principal founder. There was no mathematical research activity of consequence at NYU when Courant arrived; he set out to remedy that, and he certainly succeeded. After his death, his institute was fittingly renamed: the Courant Institute of Mathematical Sciences. I was very happy to accept the offer. I was free to do my own research, I could teach a graduate course in mathematical logic, and I would have access to an IBM 704 computer. Convinced that we would be in New York for a long time, we cut our ties to Connecticut, and moved into an apartment overlooking the Hudson River, on the upper west side of New York.

My access to the IBM 704, tempted me to see how the proof procedure Hilary and I had developed, would do on a physical computer. I was provided with two excellent colleagues, both graduate students, Donald Loveland[7] and George Logemann, to do the actual programming. The deficiencies of the Davis-Putnam proof procedure for first order logic were soon made clear. On any but the simplest problems, the memory was overwhelmed. We decided to take advantage of the availability of external storage in the form of tape drives, by changing the algorithm to make use of them We replaced Rule III from [14] (which was what would later be called binary resolution) with Rule III*, we called "splitting", which led to the divide-and-conquer algorithm for SAT that came to be called DPLL. When a split into two cases occurs, the algorithm places one of them on a stack and attends to the other.[8] The Loveland-Logemann program incorporating the DPLL algorithm for SAT was a substantial improvement over the previous attempt, but still fell far short. The paper [17], with Loveland himself one of the authors, discusses this history in more detail, and brings it up to date in connection with contemporary SAT solvers.

Our experience made it clear that any serious progress would require taming the exponential growth of substitution instances of atomic formulas. A copy of Dag Prawitz's paper [19], that arrived by postal mail at this time, contained an important clue. Although the actual proof procedure in the paper was far too unwieldy to be the basis of a useful computer program, it did highlight the significance of substitutions that make pairs of literals negations of one another. This was emphasized in my article [2] written in connection with a talk I gave at a symposium sponsored by the American Mathematical Society. Alan Robinson's resolution principle [20] that pointed to a new compelling direction, took from our work the importance of complementary literals as well input in the form of a conjunction of clauses, each consisting of a disjunction of literals. I think that is why Siekmann and Wrightson awarded my [2] a star, signifying an important article, in their anthology [21]. I discuss the history more fully in [3].

As I had hoped and rather expected, I was offered a tenure-track position at NYU. However, a better offer came from an unexpected quarter. Yeshiva College in the Washington Height neighborhood of New York City had long been offering

[7] Don later was one of my first PhD students, and, still later, a colleague.

[8] Of course the terms "divide-and-conquer" and "stack" were not yet used in computer science at that time. It may be worth mentioning that both III and III* are already in the report [15] that Hilary and I had prepared for the NSA.

an undergraduate education that combined an American liberal arts curriculum with traditional orthodox Jewish rabbinical training. However, Yeshiva College had become Yeshiva University offering secular graduate education in a number of areas. As my year as a visitor at NYU was nearing its end, I received an offer from the newly founded Graduate School of Science at Yeshiva University. It was a much more attractive offer than the one I had received from NYU. The topics of the graduate courses I would be teaching would include my own specialty. I was happy to accept. The Courant Institute graciously continued to make the IBM 704 and the talents of Loveland and Logemann available, and we were able to complete our project.

In 1965, for a number of reasons, it was time to leave Yeshiva. I returned to NYU which was to be my academic home until I retired in 1996. I was Primcipal Investigator on a mechanical theorem-proving project, but it was Don Loveland, at that time a colleague at NYU, who seized the opportunity to see his model elimination procedure implemented. The paper [16] with a number of collaborators, was the result of this effort.

In the spring of 1969, I was living in London, on sabbatical leave, when a letter arrived from Jürgen Moser, Director of the Courant Institute. A new department of computer science was being formed at Courant with my old friend and colleague Jack Schwartz as chair. The letter asked me to join, and, after some soul searching, I accepted. I found myself involved in the efforts of the new department to find its place in the Courant Institute. This did not proceed without a certain amount of friction. The department not only hoped to achieve success with cutting edge fundamental research in the Courant tradition, but also offered an undergraduate major that quickly became very popular. The need for faculty to teach these students provided us with the opportunity to hire promising new faculty.

For the academic year 1976–77, I was again on sabbatical leave. I had spent two summers in Berkeley and was eager to try a whole year. To earn a little extra money, I approached John McCarthy (who had been a fellow student at Princeton) about a summer job. He suggested that I fly out and give a talk. I thought a question that Jack Schwartz had posed about extensibility of proof checkers would be an appropriate topic. I remember working out the easy details on the plane.[9] I had an enjoyable time working at for the month of July at John's Artificial Intelligence Laboratory at Stanford University. I loved the atmosphere of play that John had fostered. The terminals that were everywhere proclaimed "Take me, I'm yours", when not in use. I was encouraged to work with the FOL proof checker that had recently been developed by Richard Weyhrauch. Using this system, I developed a complete formal proof of the pigeon-hole principle from axioms for set theory. I found it neat to be able to sit at a keyboard and actually develop a complete formal proof, but I was irritated by the need to pass through many painstaking tiny steps to justify inferences that were quite obvious. FOL formalized a "natural deduction" version of **F**irst **O**rder **L**ogic. The standard

[9] Jack and I did publish a joint paper based on this which provided a path to my Erdös number 3. There was another path via Yuri Matiyasevich.

paradigm for carrying out inferences was to strip quantifiers, apply propositional calculus, and replace quantifiers. I realized that from the viewpoint of Herbrand proofs, each of these mini-deductions could be carried out using no more than one substitution instance of each clause. I decided that this very possibility provided a reasonable characterization of what it means for an inference to be *obvious*. Using the LISP source code for the linked-conjunct theorem prover that had been developed at Bell Labs, a Stanford undergraduate successfully implemented an "obvious" facility as an add-on to FOL. I found that having this facility available, cut the length of my proof of the pigeon-hole principle by a factor of 10. This work was described at the Seventh Joint International Congress on Artificial Intelligence held in Vancouver in 1981 [4].[10]

It became the Computer Science Department's policy to provide our new hires with a review as they were completing their third year with us, to help them and us with an initial indication of their prospects for achieving tenure after their sixth year. It must have been around 1980 that I was given the task of conducting such a review of the work of Elaine Weyuker. Her PhD from Rutgers had been in theoretical computer science. Reading her recent research papers, I was surprised that she was looking, with a theoretician's eye, at a very practical problem: software testing. Programmers will certainly try to eliminate the bugs from the programs they write. But it is very difficult to envision in advance all the various environments and other circumstances in which a given program will be used. Generally, before a program is released to the public, it is tested by assembling a set of input data, and then running the program on each of these inputs. It is well understood that for both theoretical and practical reasons, running a program with a finite set of inputs can not guarantee the correctness of a program. So, "quality assurance" professionals attempt to assemble test data that they regard as "adequate". Elaine was studying this notion of adequacy.

I was both impressed by her work and intrigued by the possibility of studying in an objective manner, a notion treated in practice subjectively based on intuition and experience. We soon began a collaboration. In addition to a textbook on theoretical computer science, we wrote three joint papers. [11] is concerned with the problem of testing a program whose expected input-output behavior is not known. In such a case one couldn't tell whether the output generated by a set of test data is correct. This paper is still cited. Our [12] and [13] suggested formal definitions of adequacy based on the intuition that an adequate set of test data should separate the program being tested from all other programs with the exception of those input-output equivalent to the given program.

An article entitled *The Other Turing Machine* that appeared in the Computer Journal in 1977 caught my attention. Up to that point, Turing's name had scarcely been mentioned in historical accounts of the origin of modern digital computers. Although I was convinced that Turing's exploration of the nature of computation with his construction of a universal machine had provided their theoretical underpinnings, I had no idea how concrete the connection was. From the article I learned that Turing's Ace Report of 1946 contained the complete

[10] Parts of this paragraph were copied verbatim from my [10].

design of a stored program computer, including the electronic circuits, and even estimated its total price. In 1986, the same authors, Brian Carpenter and Doran, published a collection of some of Turing's previously unpublished manuscripts including the ACE report itself as well as the text of a remarkable lecture that Turing had delivered to the London Mathematical Society in February 1947 [1]. In the lecture, Turing explicitly tied the idea of an all-purpose stored program computer to his concept of a universal machine. In addition he provided an expansive vision of the future capabilities of the computer.

By this time, it was clear to me that the debate over whether von Neumann should share the credit for the "stored program concept" was entirely misplaced. Von Neumann who had worked on Hilbert's foundational program and who was among the first to recognize the significance of Gödel undecidability, would surely have understood the practical relevance of Turing's theoretical investigations. Also, as a logician, I could not help being aware of the historical tradition in which Turing worked. I thought that it was important that the educated public become aware of some of this. I was determined to write a book that would tell this story. I applied for and received a Guggenheim award to fund the necessary research. My "Universal Computer" was published in 2000 [6,7]. In my Introduction I wrote

> It was von Neumann's expertise as a logician and what he had learned from the English logician Alan Turing that enabled him to understand the fundamental fact that a computing machine is a logic machine. In its circuits are embodied the distilled insights of a remarkable collection of logicians, developed over centuries. Nowadays, when computer technology is advancing with such breathtaking rapidity, as we admire the truly remarkable accomplishments of the engineers, it is all too easy to overlook the logicians whose ideas made it all possible.

There was a second edition for Turing's centenary in 2012. A third edition of 2018 gave me the opportunity to write about the remarkable success of Go-playing computers using deep learning technology.

But I'm getting ahead of myself. The book was to be for the educated public. But first, I wanted to make the case for Turing's crucial role in the origin of the modern computer to fellow professionals. In my essay [5] I tried to do this, while including a brief initial section on Leibniz, so as not to neglect the historical underpinning of Turing's work.[11] Gradually Turing's role came to be recognized. By the time my book [6] appeared, I could quote Time magazine to that effect. The work of Carpenter and Doran played a crucial role in this change, as did Andrew Hodges's masterful biography of Turing. I would find it extremely gratifying to think that my essay might also have played a part.

I was surprised by an email message from Andrew Hodges, Turing's biographer, calling my attention to recent publications by the philosopher Jack

[11] In writing about Turing's work at Bletchley Park, I made the error of indicating that the Colossus was built to decrypt the Enigma traffic needed for the safety of Atlantic shipping. The Colossus was built to deal with an entirely different traffic.

Copeland that concerned Turing's introduction of the notion of an "oracle" in his Princeton dissertation. Copeland, was proposing that it was time to obtain such an oracle as a physical reality in order to be able to compute things that were uncomputable in the sense of the Church-Turing Thesis. In a popular article in the Scientific American 1999 (coauthored with Diane Proudfoot), he announced: "the search is on" for an oracle. He even proposed a capacitor storing a charge whose value was an infinite precision real number that could serve as the oracle. Anyone who read Turing's dissertation with a modicum of comprehension would have understood that Turing's oracle was a mathematical abstraction introduced for a specific mathematical purpose. And, as far as Copeland's capacitor is concerned, since the early years of the twentieth century, it has been understood that an electric charge consists of an *integer* number of electrons.

I had previously been astonished by an article by Hava Siegelmann in *Science* 1995 with the title *Computation Beyond the Turing Limit. Science* is the very prestigious journal of the American Association for the Advancement of Science, where one is used to seeing important research in the biological sciences. I was not impressed by Siegelmann's article. It was certainly true that given any language on a two-letter alphabet, she could produce one of her networks that would accept it. The secret was that the desired language was coded into an infinite precision real number which was then used as a weight in one of the "neurons" in her net. In effect the language was built into the net that accepted it. She reiterated her claim in her book, *Neural Networks and Analog Computation: Beyond the Turing Limit,* Birkhäuser, Boston 1999. I wrote about all of this in [8], and thought that was the end of the matter.

However, it seemed that there was a hypercomputation movement. There were various people who thought about computing the uncomputable, undeterred by the prospect of trying to do infinitely many things in a finite mount of time. There was quantum adiabatic cooling to solve Hilbert's tenth problem and some who were convinced that our brains are already hypercomputers. At a meeting of the American Mathematical Society in San Francisco, there was a special session on hypercomputation. I wrote two additional articles about this nonsense before saying farewell to it.

It seems to be an article of faith among theoretical computer scientists that $\mathbf{P} \neq \mathbf{NP}$. It has long seemed to me that this faith is misplaced. The heuristic arguments usually given depend on regarding $\mathbf{P}$ as the class of languages for which computationally feasible algorithms exist for deciding membership. But obviously an algorithm with cp^k as a time bound is utterly useless if c and/or k are large. In my lectures on this topic, I talk about linear programming as providing a useful lesson. Once thought to likely be $\mathbf{NP}$-hard, linear programming turned out to be in $\mathbf{P}$. Courtesy of Margaret Wright, I show an example of a large linear programming problem for which the old exponential time simplex method does better than the best "barrier" poly-time algorithm. I have no idea whether the proposition $\mathbf{P} = \mathbf{NP}$ is true. It may be that $\mathbf{P} \neq \mathbf{NP}$ is true, and that it hasn't been proved because of serious technical difficulties. But I think it is equally likely that it hasn't been proved because it is false. Perhaps there is a poly-time algorithm for SAT with a large exponent in the time bound. My experience with Hilbert's tenth problem and people's attitudes to what we were trying to do, has led me to believe that our intuitions about polynomials of high degree are not very reliable.

2012 was the year of Alan Turing's 100th birthday. So speakers who were able to give a lecture on a Turing-related subject were in considerable demand that year. I gave nine talks in places as far apart as Pisa in Italy and Arequipa in Peru. First in Ghent and then in Boston. my topic was *Universality is Ubiquitous* [9]. Turing's abstract model of computation had been able to achieve universality with just a few very rudimentary basic operations by providing his devices with unlimited memory. This suggested that the extent to which a physical device can approximate universality will depend critically on providing it with as large a memory as possible. Turing quite explicitly emphasized this in his address of 1947 [1], and it is evident in the expanding suite of things our devices can do as larger and larger memories are provided. Because so little is required of basic operations to achieve universality, it is pointless to retroactively confer universality on Babbage's analytic engine or the Colossus built in the Bletchley Park decryption effort, after imagining them provided with an infinite memory.

Do the non-coding parts of the DNA contain a computational capability? Perhaps playing a role in evolution? I shamelessly speculated along those lines.

My 90th birthday occurred in 2018. The special session on history and philosphy of computation at the meeting in Kiel of the *Computability in Europe* organization honored my birthday. I spoke on "Turing's Vision and Deep Learning". I recalled that Turing had imagined a time when a computer would have modified its original program to such an extent that the programmers would no longer understand what it was doing. Nevertheless, Turing suggested, it might be doing good work. The programmers of the neural nets that have achieved such remarkable success with the ancient game of Go find themselves in exactly this position. Likewise the programmers of self-driving cars. Thus, bringing together Turing's imaginings seventy years ago with some of the most advanced current technological achievements seems an excellent way to end this story.

References

1. Carpenter, B.E., Doran, R.W.: A. M. Turing's Ace Report of 1946 and Other Papers. MIT Press, Cambridge (1986)
2. Davis, M.: Eliminating the irrelevant from mechanical proofs. In: Proceedings of Symposia in Applied Mathematics, vol. 15, pp. 15–30 (1963). (reprinted in [21], pp. 315–330)
3. Davis, M.: The early history of automated deduction. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. 1, pp. 5–15. Elsevier, North Holland (2001)
4. Davis, M.: Obvious logical inferences. In: Proceedings of the Seventh Joint International Congress on Artificial Intelligence, pp. 530–531 (1981)

5. Davis, M.: Mathematical logic and the origin of modern computers. In: Studies in the History of Mathematics, pp. 137–165. Mathematical Association of America (1987). (reprinted in Herken, R. (ed.) The Universal Turing Machine - A Half-Century Survey, pp. 149–174. Verlag Kemmerer & Unverzagt, Oxford University Press, Hamburg (1988))
6. Davis, M.: The Universal Computer: The Road from Leibniz to Turing. W.W. Norton (2000). (Turing Centenary Edition, CRC Press, Taylor & Francis (2012). Third Edition, CRC Press, Taylor & Francis (2018))
7. Davis, M.: Engines of Logic: Mathematicians and the Origin of the Computer. W.W. Norton (2001). (paperback edition of [6])
8. Davis, M.: The myth of hypercomputation. In: Teuscher, C. (ed.) Alan Turing: Life and Legacy of a Great Thinker, pp. 195–212. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-662-05642-4_8
9. Davis, M.: Universality is ubiquitous. In: Floyd, J., Bokulich, A. (eds.) Philosophical Explorations of the Legacy of Alan Turing. BSPHS, vol. 324, pp. 153–158. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-53280-6_6
10. Davis, M.: My life as a logician. In: [18], pp. 1–33
11. Davis, M., Weyuker, E.J.: Pseudo-oracles for non-testable programs. In: ACM 1981 Conference Proceedings, pp. 254–257 (1981)
12. Davis, M., Weyuker, E.J.: A formal notion of program-based test data adequacy. Inf. Control **56**, 52–71 (1983)
13. Davis, M., Weyuker, E.J.: Metric space based test data adequacy criteria. Comput. J. **31**, 17–24 (1988)
14. Davis, M., Putnam, H.: A computing procedure for quantification theory. J. Assoc. Comput. Mach. **7**, 201–215 (1960). (reprinted in [21], pp. 125–139)
15. Davis, M., Putnam, H.: Feasible computational methods in the propositional calculus. In: [18], pp. 371–408
16. Fleisig, S., Loveland, D., Smiley, A.K., Yarmush, D.L.: An implementation of the model elimination proof procedure. J. Assoc. Comput. Mach. **21**, 124–139 (1974)
17. Loveland, D., Sabharwal, A., Selman, B.: DPLL: the core of modern satisfiability solvers. In: [18], pp. 315–335
18. Omodeo, E.G., Policriti, A. (eds.): Martin Davis on Computability, Computational Logic, and Mathematical Foundations. OCL, vol. 10. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41842-1
19. Prawitz, D.: An improved proof procedure. Theoria **26**, 102–139 (1960). (reprinted in [21], pp. 162–199)
20. Robinson, A.: A machine-oriented logic based on the resolution principle. J. Assoc. Comput. Mach. **12**, 23–41 (1965). (reprinted in [21])
21. Siekmann, J., Wrightson, G. (eds.): Automation of Reasoning, vol. 1. Springer, Heidelberg (1983). https://doi.org/10.1007/978-3-642-81955-1