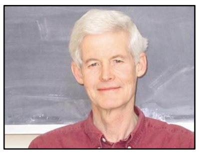
Conversations: from Alan Turing to NP-completeness

Compiled by **Jaikumar Radhakrishnan***

School of Technology and Computer Science, Tata Institute of Fundamental Research, Mumbai 400 005, India

Scientists from various fields met at the Tata Institute of Fundamental Research on 2 January 2013 to discuss Alan Turing's legacy. A panel consisting of the following made initial statements around which the discussion was conducted.

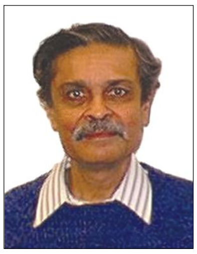


Stephen A. Cook
Department of Computer Science,
University of Toronto, Canada

Stephen Cook is among the foremost computer scientists of all times. He got his Bachelor's degree from the University of Michigan, and his Master's degree and Ph D from Harvard University. He has made deep and influential contributions to many areas, including computational complexity theory, proof complexity, programing language semantics and parallel computation. Cook is best known for his 1971 paper on 'the complexity of theorem proving procedures', where he introduced the concept of NP-completeness, and posed the famous 'P not equal to NP' question, arguably the deepest question on the foundations of efficient computation. In 1982, Cook received the ACM Turing award, the highest prize in computer science.

Manoj Gopalkrishnan
School of Technical and
Computer Science, TIFR, Mumbai

Manoj Gopalkrishnan obtained a B Tech degree from IIT Kharagpur and his Ph D from the University of Southern California. He has worked on reaction networks and self-assembly, and the role they play in the emergence of complexity and self-organization exhibited by the living system. His other interests include cognition, scientific method, computational learning theory, evolution, algorithms, computational complexity theory, physics of computation, quantum computing, algebraic geometry and category theory.



Rohit J. Parikh
Department of Computer and
Information Science, Brooklyn
College, CUNY, New York

Rohit Parikh is an eminent mathematician, logician and philosopher. He obtained his Bachelor's and Ph D degrees from Harvard University. His 1961 work on the occurrence of terminal symbols in context-free languages, now known as Parikh's theorem, is one of the corner stones of formal language theory. His current interests are in reasoning about knowledge, belief revision, game theory and philosophy of language. He has in the past worked on recursive function theory, proof theory, formal languages, nonstandard analysis and dynamic logic.

Milind Sohoni
Department of Computer Science
and Engineering, IIT Bombay,
Mumbai

Milind Sohoni obtained his Bachelor's degree from IIT Bombay, a Master's degree from the University of Illinois and his Ph D from IIT Bombay. He has worked on a variety of areas in computer science, including combinatorial optimization, geometry, game theory, formal aspects of distributed systems, etc. He is one of the co-authors of the so-called Sohoni–Mulmuley geometric complexity approach to the P versus NP problem in theoretical computer science. Sohoni is also the head of the Centre for Technology Alternatives for Rural Areas, an academic centre devoted to technology and its role in development.

Edited excerpts of the initial statements of the panelists are reproduced below.

Turing: not destined to visit India

JR: Alan Turing was destined never to visit India[1]. His elder brother, John, was born in his mother's house in

*e-mail: jaikumar@tifr.res.in

Coonoor, southern India. His mother's father was the Chief Engineer of the Madras Railway Company. Julius Turing, Alan's father, joined the Indian Civil Service in 1896, specializing in Indian Law and the Tamil language. He met his wife when he was on one of his voyages back to England. Julius Turing worked in various parts of the Madras Presidency. When the family was living in Odisha, Ethel Turing, Alan's mother, sailed back to England and on 23 June 1912, Alan Mathison Turing was born in England. Ethel and Julius Turing travelled back and forth between India and England. The boys remained in England and were educated there. In 1926, Julius Turing was superseded from the post of Chief Secretary of the Madras Presidency. He resigned and left for England. After that their connection with India was severed. As we all know, Alan Turing went to Cambridge and did very well there. He was also a champion athlete. Alan Turing's personal life, glorious and tragic, is not the subject of our discussion today; we are here to discuss various facets of the legacy that the science of computation owes to Alan Turing.

I would like to start with Rohit Parikh: Turing developed his ideas about computation not in a vacuum; there were attempts by logicians before him going back to Socrates, Hilbert, Frege, Whitehead, Russel, Brouver and several others, and finally Gödel. It will be useful to place Turing's contributions in this context of mathematical logic and philosophy.

Is *knowledge* the same as *justified true belief*?

RJP: My remarks will concern the question, 'When is knowledge justified?' This issue was raised by Socrates in his 'Dialogues with Theaetetus'. Turing addressed this issue, Gettier addressed this issue again, and Wittgenstein proposed a solution to this issue. The story goes as follows. The cast of characters is Socrates, Turing, Gettier, Wittgenstein, and a five-year-old boy.

Somebody calls at a house and a five-year old boy answers. The person calling says, 'Is your mother is at home?' The boy says, 'Sorry, she is not'. And then the person says, 'Is your father at home?' The boy says, 'Sorry, he is not here either.' Then, the person says all right can you write down a note saying Socrates called. So the boy says, 'I'll write down the note. And how do you spell Socrates?' The person says, 'S O C R A T E S'. Then there is a long pause. The boy says, 'How do you make an S?' So, there, of course, the conversation comes to an end. There is not much that can be done about this. Now what does this have to do with the other older people in our cast?

The point here is that spelling out the word 'Socrates' to the boy does not help him to write it. All of our knowledge is founded on assuming some basic facts and abilities and without them an 'explanation' is no help to us. Socrates appreciated this point as did Turing (see below for more).

While reading the 'Dialogues of Plato on Theaetetus', we came across a particular issue, namely the definition of *knowledge as justified true belief*. Now, for you to know something you have to believe it (obviously, you cannot know something you do not believe). It has to be true because knowledge must be true. The third condition is justification, namely that whatever true belief you have must be justified. In 1963, a philosopher Edmond Gettier[2] suggested that this notion that knowledge is *justified true belief*, which he mistakenly attributed to Plato, is not correct.

Gettier's objection: So his argument was the following: sometimes you can have a true belief which has justification, but there is a defect in the justification which you are not aware of. For, let us suppose that you are Mr Smith, and you are told by an authoritative source that Mr Jones will get the US Vice-President's job. So, of course, you believe it and the belief is justified because a trustworthy person told you this. Now you go into the President's office along with this man Mr Jones, and you believe that the person who is going to get the job is in that office, because Mr Jones is right there. Now it so turns out that you are the person who is going to get the job! So your belief that the person getting the job is in that office is correct. It is also justified because you heard about Mr Jones from an authoritative source. Nonetheless, it is not knowledge because your justification went through the belief that Mr Jones was getting a job, a belief that is not correct.

So Gettier, while giving this counter example, seems to suggest that this notion of knowledge as *justified true belief* is due to Plato. Now, on reading this dialogue of Plato's, my collaborator, Adriana Renero and I saw that, in fact, Socrates did not endorse this definition[3]. So this definition that knowledge is justified true belief was suggested not by Socrates, but by the boy Theaetetus. And the boy said, 'Oh yes Socrates! That is what I once heard a man say, I had forgotten but it is coming back to me. He said that true judgement with an account is knowledge. True judgement without an account falls outside knowledge.' But Socrates himself says at the end of the Dialogues: '… therefore, knowledge is neither perception nor true judgement nor an account added to true judgement.' So, quite opposite to what Gettier suggested, Socrates did not accept knowledge is *justified true belief.* But what is interesting is the reason that Socrates gave for rejecting this account. And he said the following. He said let us suppose somebody asks me for the first syllable of my name. Then I say the first syllable of my name is the letter 'S' followed by the letter 'O' and so that is the is like a explanation or justification of the syllable. But then we say what is the justification of the letter 'S'? We are reduced to a primitive element; the primitive element cannot be justified. Therefore, if this definition (of the primitive element) is correct, it cannot be known. If you cannot know the letter ‘S’, how can you know the syllable? So, Socrates gave a completely different argument from Gettier’s argument, undermining the same notion.

It turns out that Turing himself considered this issue[4]:

‘I assume then that the computation is carried out on one-dimensional paper, i.e. on a tape divided into squares. I shall also suppose that the number of symbols which may be printed is finite. If we were to allow an infinity of symbols, then there would be symbols differing to an arbitrarily small extent. The effect of this restriction of the number of symbols is not very serious. It is always possible to use sequences of symbols in the place of single symbols… . The differences from our point of view between the single and compound symbols are that the compound symbols, if they are too lengthy, cannot be observed at one glance. This is in accordance with experience. We cannot tell at a glance whether 9999999999999999 and 999999999999999 are the same.’

Then, Turing goes on further and says the following:

‘The behaviour of the computer at any moment is determined by the symbols which he is observing and his *state of mind* at that moment. We may suppose that there is a bound *B* to the number of symbols or squares which the computer can observe at one moment. If he wishes to observe more, he must use successive observations. We will also suppose that the number of states of mind which need be taken into account is finite. The reasons for this are of the same character as those which restrict the number of symbols. If we admitted an infinity of states of mind, some of them will “arbitrarily close” and will be confused. Again, the restriction is not one which seriously affects computation, since the use of more complicated states of mind can be avoided by writing more symbols on the tape.’

So what Turing points out is that behind the notion of computation, there is the notion of recognizing a single symbol. You cannot recognize a string unless you recognize a symbol and he says the alphabet must be finite for that to be possible. But Turing does not say how even when this is finite you can still recognize a symbol. Because you can see that the boy can see the letter ‘S’ but does not know what it is. The issue is that when we reduce complex problems to simpler ones, we do still have the problem of addressing the simpler problems, perhaps the simplest ones that our analysis will come up with.

Explanations come to an end somewhere: Wittgenstein also addressed the problems that arise when viewing complex tasks and ideas as being built up of simpler ones. In his example, a note is presented to a grocer asking for five red apples. Wittgenstein[5] describes how the grocer goes about honouring this request, mechanically comparing apples with other objects he knows to be red, and counting up one, two, three… . Wittgenstein says:

‘But how does he know where and how he is to look up the word “red” and what he is to do with the word “five”. Well, I assume that he acts as I have described. Explanations come to an end somewhere. But what is the meaning of the word “five”? No such thing was in question here, only how the word is used.’

Turing’s machine

JR: Parikh has already described the Turing machine, which forms the basis in the formal study of computation. I call upon Stephen Cook to explain how Turing understood and helped us understand *computation*.

SAC: Turing’s 1936 paper, ‘On computable numbers, with an application to the Entscheidungsproblem’, is probably his greatest contribution and that is why he is famous and with good reason. I do not know about whether it is a contribution to philosophy or not, but certainly he is known as the father of computer science and I believe that is justified. In his paper, Turing introduced his mathematical model of computers, which is very simple. I assume most people know what that is. But the point is modern computers had not been discovered then. Yet, he gave a convincing mathematical model of it and argued that anything a person could do following an algorithm on a piece of paper could be done using this machine. Now, he was not the first one to give a mathematical definition. Alonzo Church[6] beat him by a little bit with his lambda calculus. These notions turned out to be mathematically equivalent: a function is computable in lambda calculus if and only if it is computable by a Turing machine. But the point is that if you look at lambda calculus, the definition is totally unconvincing. Why it should compute any possible function is not clear. But Turing gave a simpler model and a convincing argument and that is why, I think, properly he should get credit.

Of course, he did more than that in his paper. He considered a version (see also MacCormick[7] on precisely what Turing proved) of the halting problem for Turing machines and proved that no Turing can solve it. The halting problem is: given a description of a Turing machine and its input, that is, the initial tape configuration, will that Turing machine halt or keep running forever. He gave a simple convincing proof, based on a diagonal argument that no Turing machine can solve this problem on all inputs. In that paper he did more. He talked about the Entsheidnungsproblem of Hilbert, which is the logical satisfiability problem for predicate calculus and showed that it was undecidable. Again, Church beat Turing by about a year, but Turing's proof was simple, for he reduced his halting problem to the Entsheidungsproblem. The reduction goes as follows. With each Turing machine one can associate a predicate calculus formula. The Turing machine runs forever precisely when the formula is satisfiable, for the infinite computation forms a model for the formula.

What is remarkable is that this simple model of Turing machine has endured now for about a century, and generally, we computer scientists are convinced that this is the correct abstraction of what is computable, and that is not going to change. And even more, it turns out that in complexity theory, where we are interested in not just what is computable but also in how hard it is to compute it, efficient computation is defined using a Turing machine. So the complexity theorists still use Turing machines.

I should also mention that Turing's reduction played an important role in my result about NP-completeness[8]. I was thinking, if we limited the time of the computation, maybe we could reduce the problem to propositional satisfiability itself instead of predicate calculus satisfiability. So, I concluded that the satisfiability problem for propositional calculus is NP-complete, and that was the first problem proved to be NP-complete. Anyway, that is the influence of Turing's work on my work, and my impression of Turing's contribution.

Physical limits of computation

JR: Turing provided us an intuitive model computation. One hears that we can apply complexity theory or ideas from computational complexity theory to real-world situations. Are the resources that are critical to the efficiency of real-life processes in any way related to the resources we use in computational complexity theory? I now call upon Manoj Gopalkrishnan, who has thought about these issues.

MG: I wanted to make a remark about something that came up in Parikh's and Cook's statements. When Turing wrote his paper on the Turing machine and the halting problem, he used the word 'computer', but he always referred to the computer as a 'he'. So what he meant by a computer was a human computer, a person performing computation. This goes back to David Hilbert's goal of trying to axiomatize the process of doing mathematics. The goal of this community of logicians was really to describe human intelligence. It was an anthropomorphic project where you wanted to come up with some axioms *to describe how the human brain works*, at least when it is engaged in doing mathematics. And that was what the Turing machine and the lambda calculus were attempting to model.

However, through the passage of time, the Church–Turing hypothesis appears to have been strengthened. My first recollection of reading something like this in print is in David Deutsch's 1985 article on quantum computing, where he elevates this to a law of physics[9] (see also Brassard[10]). He says that *one should regard the Church–Turing thesis not as a statement just about the human brain, but a statement about nature*. No matter what system you come up with in the natural world, if you try to use it as a computer, you cannot do better than a Turing machine. You cannot somehow use it to compute functions that a Turing machine cannot compute. This is the first part.

Why should we believe this? What would be a plausible argument for something like this? Certainly there are many phenomena in physics, like chaos or turbulence in fluids, that we cannot predict. So are they violating this idea that the Church–Turing thesis is a law of nature?

Well, maybe it is an unfair comparison to ask a Turing machine to predict a physical system. Because I can program a computer to make some random choices, and you cannot predict what that program is going to do without looking at the random choices that program has made either. A fair comparison is to ask a third observer to distinguish between the Turing machine and the real physical system. This is an extension of the idea of the *Turing test*[11].

So we consider a physical system and a Turing machine, and both of them are producing data. For the physical system you need some measuring instrument to interface with it and give you some measurement. That measurement has to be converted to a finite string of bits. And the Turing machine is running for some time and producing some finite string, let us say some bit sequence, so I have two bit sequences. And *if an observer can distinguish between these two* to tell which string came from the Turing machine and which from the physical system, then I would say the *Turing machine is not doing a good job of imitating the physical system*.

Why should we believe that for every choice of physical system and measuring device, we can come up with a Turing machine that can imitate it. Let us work within classical mechanics for a moment. In fact, let us stick to Newton's laws of motion. One can write down Newton's laws of motion as a system of differential equations. One might imagine breaking up this system of differential equations going on in some large space into very small cubes. If you try to figure out what each cube is doing in this space, it turns out to be following some rather simple rules. If you make each cube smaller, you can make the simulation more realistic, and it becomes harder to distinguish the computer program from the actual system. *This idea – that the behaviour of every physical system can be approximated by cellular automata – is the basis of the argument.*

Deutsch went two steps further. His next proposition was something he called the strong Church–Turing thesis. He suggested that perhaps we should start taking polynomial-time Turing machines more seriously. Sure, systems in the real world cannot compute uncomputable functions, but maybe they cannot even compute functions that are beyond polynomial time. He immediately went on to suggest that this might well be false because of quantum effects. That is an interesting story by itself, but it is more like a detail for our present purposes ... the point is one should replace the laws of classical mechanics by quantum mechanics, so one should replace Turing machines by an analogous notion, something like a quantum Turing machine. And in that world, one would still have an appropriate notion of complexity theory. So one should think of *computer science as describing laws of nature*, and not just an anthropomorphic project describing the human brain.

Before we can talk about physics and complexity theory, and their connections, we need to discuss a couple of things computer scientists have known for the last 45 years or so, but are perhaps not so well appreciated outside computer science.

How much resource is required to compute the factors of a 100 bit number? The problem is I might have a huge computer which is just a lookup table that has stored in it the factors of all 100 bit numbers. All I have to do in this computer is go and read the answers. This is unsatisfactory; such a computer would have a size more than 2^{100}, and is impractical. So *notions of uniformity* come in. I do not allow my computer to be just a lookup table. I insist either that it be a Turing machine, or in a less uniform model, but still keeping some uniformity. I insist that it is a circuit, so the computation is uniform for an exponentially large number of inputs. But if I change the size of the input, I am allowed to use a different computer. A physical way of interpreting this might be that we are accounting not just for the cost of computation, but also for the cost of building the computer.

There is another issue which is *speedup theorems*. Complexity theory does not measure time in seconds. All we can say is that time grows asymptotically and there is some scaling, and the reason for this is that we have the linear speedup theorem. We can pump up the alphabet size and do things twice as fast as we could do it before. It seems to me that this is partly a cheat. We cannot really do this in the real world. If we are building a computer with a wider bus, we are saving on time, but then we are spending more energy. One should really think more carefully about not just how much time is required for it, but also how much energy is required for it. Maybe that will also give some insight into settings in the natural world where one wants to apply ideas from computer science, viewing it as a law of physics. So one wants to take a computational view to natural phenomena.

So time by itself is not the right measure. I do not think energy is the right measure either. The problem with energy actually goes back to some work of Charles Bennett and Rolf Landauer[12], motivated by a question from John von Neumann. Von Neumann was interested in this *relationship between thermodynamics and computers*. He wanted to imagine an analogy of the computer with the steam engine. A computer takes in energy and produces not work as we understand it, but some sort of computational work. He wanted to say that there must be a limit, that one needs a certain amount of energy to do computation.

There were subtleties which came out in the work of Bennett and Landauer. The conventional interpretation for their results is that no energy is required for computation; there is no lower bound. But a more careful reading of their work shows that their arguments are in the equilibrium thermodynamic limit, which immediately forces them to do their computations infinitely slowly, in the infinite time limit. So their results do not really apply to the *finite time limit*. Nevertheless, their results also seem to rule out energy by itself as a complexity measure. Perhaps what I would suggest is some sort of product of energy and time as the right quantity to look at. I have some ideas on this, but they would be too speculative to discuss at the moment.

The P ≠ NP question and geometric invariant theory

JR: The P ≠ NP question has been a central question in complexity theory ever since Cook's paper[12]. There are several versions of this problem. One version asks whether, given a mathematical statement with a short proof, one can find that proof efficiently (in time polynomial in the length of the final proof). Another version asks if, given a system of polynomial equations in several variables, where each polynomial involves at most three variables, it is possible to efficiently determine if there is 0–1 solution to this system of equations. Ketan Mulmuley and Milind Sohoni have a novel approach to this problem. I would like to call upon Milind Sohoni to summarize for us the geometric invariant theory approach and how it connects this question to wider questions in mathematics.

MS: One can view the Cook–Levin theorem or even Turing's theory as a way of representing computation, or algebraizing it into a static object, which has really no element of time. We are trying to show that some problem G cannot be reduced to another problem F with limited resources. Our approach, based on geometric invariant theory, actually looks at transformations between problems as also static object. So, the transformation as an algorithm has been converted into some sort of algebraic structure. So, we consider all functions reducible to F, which we call orbit of F. This orbit is an algebraic object obtained by the action on F by a suitable group. So, the real question is whether G is in the orbit of F or not, and in interesting specific cases, we would like to show that G is not in the orbit of F. In joint work with Ketan Mulmuley, we place this problem in the framework of representation theory, which also plays a central role in many problems in physics, and links them to some hard problems in mathematics. We are trying to explicitly describe invariants that must hold for elements of the orbit of F, which we can show that G does not have. We are studying the nature of these invariants.

As a concrete example, consider an $n \times n$ matrix A of variables ($X_{ij} : i, j = 1, 2, \ldots, n$). There are two important functions that can be defined using these variables, the permanent and determinant. We would like to show that the permanent is a harder function than the determinant. For our purposes it would be sufficient if we can show that there is no linear transformation that transforms a permanent into a determinant, that is, we can build a small matrix B, whose entries are just linear forms in the entries of A, so that the permanent of B is the determinant of A. In order for this question to be amenable to tools from geometric invariant theory, we first need to show that the determinant and permanent are stable functions. This we have managed to do. Using this we hope to obtain geometric invariants to explain using invariants and representations why the permanent and determinant are fundamentally different. One could conclude that the permanent function, which is central to counting the number of solutions to NP-hard problems cannot be computed in polynomial time.

This approach relates these problems in computational complexity theory to several classical problems in other areas. For example, one important problem in mathematics that this relates to is the plethysm problem, which attempts to understand tensor products of basic representations and their factorization.

This approach requires further work in the following directions: (i) development of suitable mathematical theories to enable careful algebraic accounting, such as is achieved through quantum groups, and (ii) through deeper understanding of proof techniques in geometric complexity theory.

1. Sriram, V., Origins of the father of computer science. *The Hindu* (Madras and the World), 20 August 2012; http://www.thehindu.com/news/cities/chennai/article3796302.ece
2. Gettier, E., Is justified true belief knowledge? *Analysis*, 1963, **23**, 121–123.
3. Parikh, R. and Renerao, A., Justified true belief: Plato, Gettier and Turing. In *Turing 100: Philosophical Explorations of the Legacy of Alan Turing* (eds Bokulich, A. and Floyd, J.), Boston Studies in the Philosophy of Science, Springer Verlag, forthcoming.
4. Turing, A. M., On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc., Ser. 2*, 1936, **42**, 230–265.
5. Wittgenstein, L., *Philosophical Investigations*, Blackwell Publishing, 2001.
6. Church, A., A note on the Entscheidungsproblem. *J. Symb. Logic*, 1936, **1**, 40–41.
7. MacCormick, J., On computable numbers, with an application to the Entscheidungsproblem, July 2010; users.dickinson.edu/~jmac/selected-talks/turing-and-halting-problem.pdf
8. Cook, S. A., The complexity of theorem-proving procedures. In Symposium on Theory of Computing, 1971, pp. 151–158.
9. Deutsch, D., Quantum theory, the Church–Turing principle and the universal quantum computer. *Proc. R. Soc. London, Ser. A*, 1985, **400**(1818), 97–117.
10. Brassard, G., Is information the key? *Nature Phys.*, 2005, **1**, 2–4; doi: 10.1038/nphys134.
11. Turing, A., Computing machinery and intelligence. *Mind*, 1950, **59**(236), 433–460.
12. Bennett, C. H. and Landauer, R., The fundamental physical limits of computation. *Sci. Am.*, 1985, **253**(1), 48–56.

ACKNOWLEDGEMENT. We are grateful to John Barretto (Tata Institute of Fundamental Research) for his help in transcribing the original recording of the discussion, on which the above is based.