Stephen Cook
1982 ACM Turing Award recipient
Interviewed by Bruce Kapron
February 25, 2016

BK = Bruce Kapron (Interviewer)
SC = Stephen Cook (A.M. Turing Recipient)

BK: Hello, this is Bruce Kapron. It is February the 25th, 2016 and this is a recording of an interview that I had with Professor Steve Cook at the University of Toronto, where Steve is University Professor in the Computer Science and Mathematics Departments. This is part of the ACM Turing Award winners' project. Professor Cook received the ACM Turing Award in 1982 in recognition for his contributions to the theory of computational complexity, and in particular the theory of NP-completeness, which he introduced in his 1971 paper "The Complexity of Theorem-Proving Procedures." So now what you'll be seeing is my interview with him.

SC: Oh. Steve, you were… Hello.

BK: You were born and you grew up not far from Toronto in Western New York. As a child, did you already have an interest in science or mathematics?

SC: Yeah. Well, I certainly had an interest in science. My father was a chemist and worked for a Union Carbide branch, so I was interested in science. I was good in mathematics, but I didn't think in high school that that was going to be my chief interest.

But the big influence on me when I was in high school was Wilson Greatbatch, who was a resident of Clarence, New York, same as me. He was an electrical engineer, but a very creative one. Transistors were a very new thing then, and he designed a transistor circuit that went "Bip, bip, bip," and eventually turned it into an artificial pacemaker for hearts which was implantable. That had never been done before. Well, of course you couldn't do it with vacuum tubes, obviously. So he eventually got ushered into the Inventors Hall of Fame in the United States for inventing this thing.

But while he was doing this and I was in high school, I helped him out. He had a little shop in the top of his garage in which he worked. He would draw pictures of

transistorized circuits and I would solder them together. Then he'd try them out and I could see on an oscilloscope this "Bip, bip, bip."

BK: This was the late 1950s?

SC: That's right. That's right, because I was in high school. So like I graduated from high school about 1957. Then I don't think it actually got turned into a pacemaker till some years after that. Also, he worked for an electronic firm and he got me summer jobs there. So I was very interested in electronics and I thought that was going to be my profession.

BK: You mentioned your father. And your mother was also a teacher?

SC: Yeah. Well, yes.

BK: She was a teacher, yes?

SC: Yes. Well, she had two master's degrees. My father and mother met at the University of Michigan. She got a master's degree in English and history then I guess. Yeah. And of course where my father got his PhD in chemistry. Then later on when… I have three brothers and she was mostly taking care of us, but when we were all off, going off to various places, then she was a teacher. Oh, she got a second master's degree in English and then she was teaching at a community college in the area.

BK: So you mentioned that you had quite an early introduction to I guess at the time what was very high technology electronics. What was your first exposure to computers and computer programming?

SC: Yes. Even in high school, I had a good math teacher who was somewhat interested in computers, and he took us to downtown Buffalo to some meeting. So I already had some idea about computers. But it really wasn't till I went to University of Michigan, which was my undergraduate school… Since both my parents were alumni, there seemed to be no choice. My very first year, which would have been I guess '58, the spring term, I took a course, a one-hour course in programming. This was from Bernard Galler. We learned how to program, and I think it was the IBM 650, which was a vacuum-tube machine whose memory was on a drum, just to put you in the category of what computers were like then.

That was probably my real access to computers. But I enjoyed it a lot and I started writing fun programs like… I remember early on, I wrote a program to test Goldbach's conjecture, which was that every even integer is the sum of two primes. So I tested it up to some large number and it turned out to be verified.

BK: And at Michigan, you started off in electrical engineering?

SC: That's right, I actually start-… It was engineering science I think, but I was thinking in terms of electrical engineering. But my very first year I took a calculus course from Nicholas Kazarinoff. It was I guess not just the completely typical calculus course, slightly more advanced. Anyway, I got quite interested in the course and Kazarinoff got interested in me. He would give me special homework theorems to work on. In fact, he was so impressed that he said I could skip the second-year calculus course – which I kind of regret now because I never learned two-variable calculus – and go on… there was a third-year course. And also then I took a third-year course in linear algebra. So yeah, I got off to a good start in mathematics.

BK: And eventually you switched into…?

SC: That's right. After… I think I took two and a half years, and then I finally switched out of engineering and became an official math major. That's right.

BK: So you were successful enough that you ended up going to Harvard for your graduate work.

SC: Yes, indeed. Yeah, yeah. I was quite excited about that. I must have gotten good reference letters. So yes, I guess it would have been '61 I joined, I became a student for a master's degree in the math department. I got that in '62.

Then I needed a thesis advisor, but I'd taken a logic course from Hao Wang. He was not in the math department. He was in the Division of Applied Physics. But I mean he was really a logician. Logic was his major interest. But he also was interested in computers. And before coming to Harvard, he worked I guess for both Bell Telephone Labs and the IBM Watson Research Laboratory in Yorktown Heights, New York. There he wrote a program to prove theorems in propositional calculus automatically. That was one of his interests, trying to make automatic theorem provers. It apparently was very successful because he used an IBM 704, which is another vacuum-tube machine, and in just three minutes it proved all the hundred or so propositional tautologies in Russell and Whitehead's Principia Mathematica. That was a big deal because other people, Shaw and Simon had tried to make automatic theorem provers for propositional calculus and apparently they weren't very successful.

BK: Professor Wang was also very interested in foundational issues in mathematics in logic and computability. I'm wondering… It seems that both with the automated theorem proving and with his interest in computability, that really had a big influence on a lot of the work that you've done.

SC: That's right. I mean it certainly did. There was no question about that, that yeah, he was interested in the *Entscheidungsproblem* of Hilbert, which is the problem of determining whether a given predicate calculus formula is satisfiable or not. Then of course Turing and others proved that it's unsatisfiable. So then how simple a

formula, class of formulas can you make to make it still unsatisfiable? And there was the so-called AEA case, for all exists for all case, and then proved it was still unsatisfiable. So he was part of that. And yes, that did have an influence on me later on.

BK: But computational complexity has always been a theme in your work. I'm wondering if you can say a little bit more about what your PhD research was based on.

SC: Ah! Yeah, okay. So some background for that. Computational complexity was a new subject of course, and this was in the early 1960s. In '63 I think it was, Hartmanis and Stearns published their famous paper in which they introduced the word "computational complexity" on "The Computational Complexity of Functions" or something like that. So Hartmanis came to Harvard and gave a talk, and so I became quite interested in that.

Then the other influence was by Alan Cobham. Alan Cobham had a degree from Harvard and he was a graduate student there. He wrote a thesis, but he never got his PhD because the math department, in addition to a thesis, it requires a minor thesis and Alan never bothered with a minor thesis. But he went off to work for, again, IBM Watson Research Lab. He was connected with… He was a friend of Hao Wang and he would come, so I got to know him. His famous paper was "The Intrinsic Computational Difficulty of Functions." That was his paper he wrote. He was interested in… I think this was really before Hartmanis–Stearns, yeah, that happened, but he was interested in "In what sense are some computational problems harder than others?" and he compared multiplication and addition as an example, "In what way is multiplication harder than addition?" But his result, which…

I mean one thing he did that really had an influence was he introduced the notion of what we would now call polynomial-time computable functions. He argued that this is an interesting class, a complexity class, because it seems to be all feasible functions, and if you're not polynomial-time computable, you're not going to be feasible. He made that argument. Then he also came up with an interesting characterization of the polynomial-time computable functions. Namely a function is polynomial-time computable if and only if it can be obtained from certain initial functions and applying the operations of composition and limited recursion on notation. And that's something he made up, a new kind of recursion. Primitive recursion of course had been well known for many years, but to characterize the poly-time computable functions, you needed this other notion, limited recursion on notation. So that also had an influ-… I was very interested in that.

BK: But in your PhD, you considered the question of the difficulty of multiplication.

SC: That is correct, yes. Maybe that was because of Alan Cobham's question. Yes, of course. It's very hard to prove lower bounds. We all know that now and that was

certainly true then. The only way you'd get a lower bound on multiplication was to look at the case where the inputs are only read… What do you call that? So the inputs are restricted so that after each digit is given, you'll have to give the output for the multiplication.

BK: Oh, it's…

SC: Online. I'm just thinking online. Sorry for the… Yeah. So this was an online model of a Turing machine, which you think of an unlimited number of digits. You're multiplying two numbers in decimal or binary, whatever, it doesn't matter. So the first *n*, string of *n* digits would be the *n* least significant digits in the two numbers, and that's enough information to get their product. So you had to output their product and then go on to the next two digits and then keep outputting the product and so on. That was the online model. In that model, I was able to get a lower bound of *n* log *n* over… just over *n* log *n*. What was it? I can't remember exactly. *n* log *n* over log log *n* I think it was. Yeah. So that was a non-trivial lower bound. That was part of my thesis. The other parts again talked about computational, number theoretic, or real function problems.

BK: After graduating, you became a faculty member in the math department at the University of California, Berkeley.

SC: That's right.

BK: What year was that?

SC: That was 1966. Yeah, so I finished my PhD in 1966. I had been offered a job. Now the job was half mathematics in the math department and half in the… It wasn't a computer science department, although they did have a budding computer science department just starting. It was a research job in the Computer Center or something, something like that, so it was only half in mathematics. So yes. And so I started that out in the fall of 1966.

BK: So you were in Berkeley in the late '60s, which must have been an interesting time intellectually and culturally as well.

SC: Indeed. Of course yeah. This definitely was… The free speech movement was in full form and there were crowds of people there. In some cases, the police were called in. There was teargas and all kinds of stuff. Yes, all that was going on in the '60s. Yes.

BK: As soon as you got there, I assume… or even, as you say, working on your PhD you were already thinking of a lot of questions about computational complexity and even thinking about polynomial time, so…

SC: Yes. Well, of course yeah, the polynomial time definitely came from Cobham, so I was interested in that. Early on, I did circulate a mimeographed set of notes on something like classes of primitive recursive functions, so there were complexity classes of primitive recursive functions. So there was work at Princeton at the time. There was a logician at Princeton, and I had read parts of Bennett's thesis. Anyway, his thesis was called *On Spectra*, which was actually a predicate calculus sign, but it had a lot of complexity theory results based on logical…

So there was kind of two different, yeah, classes of people then. I mean these were the logicians doing complexity theory. So he didn't talk about Turing machines ever that I remember, but I remember one of the complexity classes he talked about were the extended positive rudimentary relations, the class of extended positive rudimentary relations, which had some fancy definition, logical definition. Then I eventually realized, "You know what this is? This is nondeterministic polynomial time." The same characterization. So that's probably the first time I became interested in what we now call "NP."

BK: And what year was that?

SC: That would have been, well, '67, because I circulated this stuff, it has the date on it, 1967. So…

BK: So then you were already aware of the question of what the relationship would be between P and NP?

SC: Well, absolutely. I even put it out in there that "Oh, this is interesting." So one assumes that there are problems in nondeterministic poly time that can't be done in deterministic polynomial time, but might be aware they're hard to prove that. And I turned out to be right on that point.

BK: And while you were at Berkeley, you were already doing work that then ended up at the ACM's Symposium on Theory of Computing…

SC: Yeah, that's right. I had a couple ACM STOC conference papers. One of them was this characterization, another characterization of polynomial time, which I think is quite intriguing. So a problem can be solved in deterministic polynomial time if and only if it can be solved by a so-called auxiliary logspace pushdown machine. So what is this? That's a Turing machine that has read/write input tape and it has a work tape where they commonly use log-space, order log-space symbols on its work tape and do the work. But then… So that model is a characterization of what we call "logspace," which is, as far as we know, a proper subset of polynomial time. Of course we can't prove that either, but we *assume* logspace is certainly a subset of polynomial time, but… we assume it's a proper subset. But if you add the pushdown stack, I proved you exactly get polynomial time. And not only that, the nondeterministic version also gave you deterministic

polynomial time. So I thought that was kind of a neat result. That got published in the ACM journal.

BK: So you left Berkeley and you came to University of Toronto in 1970. Were you recruited by people at Toronto?

SC: The story there is that the math department denied me tenure, because otherwise I would have stayed certainly, especially since I had just married my wife Linda, who was raised in Berkeley, the centre of universe, and she had no desire to leave Berkeley. So this was quite… I should say, to be fair, when I proposed to her, I told her I might not get tenure. So she was forewarned.

In any case, yeah, so it looked like I had to get a job somewhere else. So I applied at other places, including Yale and University of Washington and I think IBM Research. Then I went on a trip. But the Toronto connection, I didn't have any sense of going to Toronto, even though I have to say that I had no problem with Canada because we lived in Buffalo, we used to go to resorts, the summer resorts in Ontario, so that's fine. But anyway, I didn't think of Toronto. But yeah, in fact I was recruited. Somebody in the computer science department at Berkeley who had just come from… who had been at U of T. He was Canadian but he was recruited by the Berkeley computer science department. So anyway, he called Tom Hull, the chair of the computer science department, and gave my name and said, "Maybe you should look into this." Yeah, so Tom eventually got hold of me, not on that recruiting trip but later on. So Linda and I went to Toronto and eventually thought, "Oh, this is a great place." Actually it was a very good budding department.

BK: It must have been young at the time.

SC: Yeah, it was quite new, but they still had a number of people. Tom Hull had just moved there, but Kelly Gotlieb was there and Patt Hume. And in fact before I came, Allan Borodin who is now quite well known as a complexity theorist and also Derek Corneil was there, and he is a graph theorist, well known. So it was clearly an up and coming department. Yeah, so certainly it turned out to be a very good choice.

BK: And making the switch to computer science seemed like a natural…?

SC: Oh, that's true. But the people who recruited me were all in computer science. Actually… So my first appointment was half in math and half in computer science, and then yes, I quickly switched to computer science, realizing the grass was greener in the computer science department, yes.

BK: You presented "The Complexity of Theorem-Proving Procedures" at ACM STOC in 1971, and that's the paper that introduces what we now call the theory of NP-completeness. How was your paper received at the time?

SC: Actually it was received very well. Yeah, I gave it to a large audience. I gave the talk to a large audience and there were people there. I remember Michael Rabin was one and he seemed quite impressed. He had been thinking along similar lines actually. So I got positive feedback from that paper, for sure.

BK: Can you briefly describe a little bit about what's in the paper, since it's not in the form of NP-completeness that we know today?

SC: For sure. Not at all. Those symbols were never uttered in my paper. So here's the story. I was interested in the complexity of theorem proving. In fact, that's the name of the paper, right? "The Complexity of Theorem-Proving Procedures," which doesn't sound very much like "NP-completeness." So when I submitted the paper to STOC, I actually didn't have my result there. I had a section on propositional calculus and complexity, and I had a section on predicate calculus, but I didn't have any really big results.

But after they accepted my paper despite this… Because STOC was much easier to get into in those days than now. Its standards have gone way up. But then when I started thinking about writing the final version, I had this idea of completeness, of complete problems. And of course where did the idea come from? It came from completeness for recursively enumerable sets, and in fact the – what is it? – the unsatisfiable predicate calculus formulas are complete for recursively enumerable problems. I knew that and my advisor was very interested in that, so I credit him for giving me the idea "Well, why can't we do this at a lower level for propositional formulas?" and then the analogue of recursively enumerable becomes nondeterministic polynomial time. And then I proved that the… Well, what I actually proved was that the valid propositional tautology, validity of a propositional tautology – which is in co-NP, it's not in NP – was complete for this class. But the reductions I put in that paper were not the many-one reductions that Karp used and I use now, but they were Turing, polynomial-time Turing reductions, which are much more general reductions.

Yeah, so I didn't have the words "NP" and "P" – that was due to Dick Karp later – and I didn't have the same reduction, I had a more general reduction, and I had only three complete problems. And of course Dick Karp later, a year later had 21. I did have three in there. Of course there was the tautologies and subgraph isomorphism – given two graphs, is the first one isomorphic to a subgraph of the other? – and then three… Oh. And then, well, of course you could also instead of tautologies in general, you could look tautologies in disjunctive normal form, that was complete. And also, I did also have three DNFs. So conjunctive normal form with just three literals in a conjunct, that was also NP-complete, so that's what I had in my paper.

BK: And was there a conjecture in there about primality?

SC: I mentioned other possibilities. Yes, primality testing. I said that's a candidate maybe for completeness. I should look and see whether I was doubtful, because in

fact there are randomized algorithms for primes. I may have said that, I don't remember. And the other, an open question was graph isomorphism. Neither of those are thought to be NP-complete now. In fact, primes are in poly time. So yeah, I did mention those. And of course graph isomorphism is definitely thought not to be NP-complete, although nobody's gotten a poly-time algorithm for it.

BK: You mentioned Richard Karp's paper following onto yours. How long did it take for researchers to sort of realize the significance of NP-completeness?

SC: Oh, I think it came very fast. I mean it was obvious, and Karp's paper was very well written. He had 21 examples of NP-complete problem. He cleaned up the terminology. He introduced "P" for poly-time for "NP" for nondeterministic polynomial time. That was new notation, very clean. He also introduced "many-one poly-time reducibility" whereas I had the more general kind of Turing reducibility, poly-time Turing reducibility. And all those were very clean, nice definitions. So that paper definitely caught on very rapidly.

BK: So in the 1970s, there was explosion of research. Were you surprised at the impact?

SC: Yeah, I would say I hadn't quite anticipated that there were so many NP-complete problems. You know, I think there were two kinds of people working in this field. There were the logicians and the people… algorithmic guy. And Karp was definitely an algorithmic guy. My training had all been with the logicians, so I think that's my excuse for not realizing all of these examples of NP-completeness. And I should say on this term, a similar example for the notion of polynomial time of course was independently introduced by Jack Edmonds, about the same time as Alan Cobham wrote his paper. But they were from two different areas – Cobham was a logician and Jack was an algorithms guy. I mean they're two different fields, and so quite independently they came up with polynomial time. But I didn't know about Jack's stuff at all until much later. Cobham was my source for polynomial time.

BK: The question of whether P is equal to NP is one that's intrigued and frustrated a lot of researchers, so I have to ask you, what's your opinion on the status of P versus NP?

SC: Oh. Well, that's easy. I think P not equal to NP, and I think a majority of complexity theorists believe it. Well, so here's my tune on this. First of all, we're really good at finding algorithms for things. I mean there's a whole algorithms course we teach undergraduates in all these methods of finding algorithms, and lots and lots of examples. But for lower bounds, we aren't good at finding lower bounds. And here's my proof. If you look at the sequence of complexity-classes log-space, which is a subset of polynomial time P, which is a subset of NP, nondeterministic poly-time, which is a subset of polynomial space. So here we have a sequence of three inclusions starting from log-space and ending in

polynomial time. There's an easy proof that log-space is a proper subset of polynomial space just by diagonalization. Therefore one of those intermediate three inclusions has to be proper. We can't prove any of them are proper. QED.

BK: But I guess even in 1967 now, you were telling us that your feeling was that P is not equal to NP.

SC: That's right. Based on attempts really seemed much harder to solve NP-hard problems, or NP problems in general. So yeah, I guess I conjectured that way back then. Yes.

BK: So not only is that problem P versus NP's central problem or the central problem in theoretically computer science, with the introduction of the Millennium Problems, Millennium by the Clay Foundation, the seven problems, it sort of has been acknowledged as one of the most open problems…

SC: Yeah, I guess one of the most interesting open prob-… important, important open problems, yes.

BK: So were you consulted about that inclusion, or did you just…?

SC: No, I was not consulted about the inclusion, but after they decided it, they did ask me to make a write-up for it. So I did contribute a write-up for that question, for the background.

BK: And I guess before that, Professor Smale had already… he had a list of 21 problems I think and included it.

SC: Oh, I guess he did. Yeah, that's right. He had already listed it as one of the… So definitely there's a consensus this is an important problem.

BK: What about the bigger significance? I mean it even gets into popular culture in *The Simpsons* or whatever. So how do you feel about…

SC: [laughs]

BK: …I guess a little bit what's the real significance of the P versus NP problem?

SC: Well, I don't know. Yeah, you're right, it gets into… I guess everybody, many people know P and NP and they don't understand what it is. Well, obviously it's an important question. And, well, for one thing, if P equals NP, it's going to rule out a lot of cryptography. It's hard to imagine how we could have any of the cryptographic protocols like RSA and so on. Public key encryption seemed to be impossible. So that's on the one hand if it turns out P equals NP. And on the other hand, if P not equal to NP, of course you want to know more, you want to know just how hard is NP and so on. You can't help but learning a lot more about the

problems. Either way. Either way it goes, P equals NP or P not equal to NP, we're going to learn a lot more about computation if the problem is solved.

BK: Right from the start of your research career, I guess what you'd call proof complexity has been a central focus. I think many computer scientists know about computational complexity, but maybe not so many know about proof complexity. So I'm wondering if you could describe a little bit what the concerns are and what some of the basic questions…

SC: Well, of course proof complexity in the propositional form is quite related to the P-NP question because you want to know… I mean a good aspect of proof complexity is you look at proof systems for proving tautologies, for example, which is equivalent to proving negations are unsatisfiable. So we have lots of standard proof systems for doing this.

Now the issue there is though "How long is the shortest proof? Can you get an upper bound on the length of the shortest proof in some proof system?" The conjecture there is there's no polynomial upper bound no matter what proof system, you know, efficient proof system. Under a reasonable definition of proof system, could you get a polynomial upper bound on the length of every tautology, a polynomial in the length of the tautology of course? So we conjecture, no, that's sort of like "P not equal to NP" conjecture. I mean that sort of would imply P not equal to NP. But the… So you're… If there were such things, what I'm trying to say, if there were a proof system, efficient proof system to get poly-time proofs to all tautologies, then NP would equal co-NP, meaning that the complement of any NP problem would also be in NP. And again, there's lots and lots of examples, and we just don't think that's true. So we conjecture…

So this is all… proof complexity is certainly tied up with computational complexity. That's just one example.

BK: And you mentioned Professor Wang's program for proving the propositional tautologies in *Principia*, and these days both automated theorem proving or satisfiability solving in the propositional case has really become an important technology.

SC: Absolutely. Yeah, of course. And what was surprising is yeah, there are theorem provers especially for sets of clauses, for propositional problems that are in conjunctive normal form that are incredibly efficient. I mean they can find proofs or disproofs for even tens of thousands of clauses. So that's an interesting, very interesting, important subject in its own right. Of course, despite their great successes for some cases, you can always stump them…

BK: Of course.

SC: …by coming up with hard examples. Like the pigeonhole tautologies is a good example.

BK: Besides the work in computational complexity, you've worked in other areas including parallel computing and theory of programming languages. Is there a theme that goes through all this work that holds it together?

SC: Well, it's all… I mean certainly the parallel computing part is an interesting… it's certainly mainstream, interesting complexity theory, because in practice now, computers have many, many processors and you're very interested in how much time you save if you have a whole bunch of processors. And there are conjectures of course… Well, for some problems you can't save much of any time. For other problems, you can save time hugely. So that's a very important and natural complexity question for parallel. What was the other…?

BK: The other one was theory of programming languages.

SC: Oh yeah. Well, that's Hoare. My contribution there was to Hoare logic.

BK: Right. And I think I've heard that referred to as "Cook's theorem of the relative completeness of Hoare's logic."

SC: [laughs]

BK: And I was quite surprised when I heard it referred to as "Cook's theorem."

SC: Well, apparently it's had some effect. I mean that's really not my field, programming langua-… It's a very important field, proving correctness of programs of course, and Hoare had this logic, a whole method of proving correctness. By putting in certain kinds of assertions here and there, you could use it to prove correctness of programs. So I succeeded that in a certain sense his logic was complete, so you can always do it if it's correct. So yeah, that was just one paper, but it seemed to have made a bit of a difference.

BK: Despite all the advances in computing technology – and here I have '71, but now I should say '67, which is when you originally formulated the problem – you know, the ideas about P and NP are still central questions in computer science. I'm wondering why they have such enduring significance.

SC: Well, I mean I guess – I tried to answer that before, right? – because they're obviously important, they're very important questions, and whichever way they resolve, we're going to learn a lot more about computation. And of course it's especially relevant at cryptographic protocols, which really would be much more difficult if it turns out to P equal NP.

BK: Apart from your research, you also have a lot of impact in terms of teaching and graduate supervision. I don't know the count now, but you've had over 30 PhD students.

SC: Yeah, it's 33 I think. 33, 34 graduate students. Some of them co-supervised I have to say. Especially later, in my more recent years, I have to credit other people like Toni Pitassi for being really good and helpful co-supervisors for students. So indeed, yes.

BK: And I'm wondering how you see how those three, teaching and supervision and research, interact with each other.

SC: Yeah. Well, there's no question supervising graduate students interacts with research, because your graduate students become co-authors in the papers. I mean it's extremely enlightening and important to have graduate students. I'm very grateful to my graduate students, including yourself because we have a joint paper. There's no question that working with graduate students is very enlightening and rewarding. And yeah, I also teach classes to undergraduates and, no, I enjoy that as long as I don't teach too many. Especially if I'm teaching in the areas I really like, which are complexity theory and logic. Those are my two favourite subjects, and I do enjoy teaching those courses because I think they're really neat and I try to get the students to agree.

BK: Well, I have to say from my personal experience that I think from the first time I walked into your office, you just treated me more as a colleague than a student, and I think it all was driven by real interest in the research questions.

SC: Absolutely, yeah. Well, you're definitely a helpful colleague indeed.

BK: So you're known to be an avid sailor. When did you first start sailing?

SC: Well, the first serious time I started sailing is when I went to Berkeley. I had tried to sail a few times. Even at Harvard, you sail. The sailing there on the river isn't too good, so I decided not to. But when I went to Berkeley, then you could stand up on the hill and see the San Francisco Bay opening up, the high road opening up for you, and all the boats out there really looked like fun. And I said, "Okay, I'm going to learn to sail." So I joined the student sailing club, UC- whatever it is. UCYC, University of California Yacht Club. But they let faculty members in. So then I joined and they had a very nice teaching program on Sundays. So you went to Sundays and you got in a boat and you had a teacher, and it was all very reasonably priced and they would teach you how to sail. So I really took to that right away and became a good sailor.

And the other thing is I met my wife, because she was secretary. She was an undergraduate. She was secretary of the club. So that was a great thing. Two good reasons why I learned to sail. But yes, I absolutely enjoyed it.

BK: You're a member of the Royal Canadian Yacht Club.

SC: I am. Yes, that's true. And so yeah, Linda and I are members of the Royal Canadian Yacht Club here. That was a recruiting tool by Tom Hull by the way to come to Toronto. He pointed out, "We do have a lake here. We don't have the ocean, but we do have Lake Ontario and they do sail." And he invited a member of the RCYC to lunch to explain how good it was. So indeed.

BK: And you raced for a long time.

SC: I still do.

BK: You still do?

SC: I still do. Yes, from the very beginning I took to sailboat racing, and I still do. Right now I sail… The club owns a fleet of… called Ideal 18s, two-person keelboats that are easy to sail. But they're all alike and we all go out and race. I have races Thursday evenings and sometimes on Saturdays. I still do that and I get a kick out of it.

BK: You mentioned your wife Linda. You still live with Linda in Toronto. And for a long time, she worked at University of Toronto, is that correct?

SC: Yeah. She worked at the admissions office at first and then at the registrar's office at University College. Then we started having children, and after a while we realized that it was tough, it was better if she quit. So she eventually quit and stayed home.

BK: And you have two sons, and I guess both of them have followed in your footsteps in their own way. I wonder if you could tell us a little bit more about them.

SC: Well, Gordon is the older one, born in 1978. And he took to sailing very early. Well, we encouraged him of course, Linda and I, in particular sailing in the Optimist dinghy, which is for children 15 and under, and they have a world-class racing… It's a world-class racing boat. Optimist dinghies, they're… What? They're eight feet long and not very big and have one sail, and they're for children. But they have international regattas. Anyway, Gordon took to that right away and practised and raced in the local races. Eventually he competed to represent Canada in the Optimist Worlds, and only five boats are allowed from each country. So he did that and he actually did that twice, and once it was in Greece and the second time it was in Spain, and I got to go and it was great.

Then he graduated. He got his engineering degree from Queen's University and started a little company making carbon-fibre stuff. But then he really liked sailing, so he decided to try out for the Olympics. So he worked very hard to… This is in

the 49er class, is one of the Olympic-class racing boats, a 49er. It's about 14-18 feet long and very hot, very fast. So that was Gordon's choice, and he worked very, very hard and eventually competed for Canada in two different Olympics. In 2008 it was in China and 2012 it was in England.

BK: And James.

SC: And James. James also started out sailing, but he didn't take to it quite in the same way. But he was certainly interested in computers from age 3 on and also mathematics. So that was his thing. So he got his undergraduate degree at University of Toronto in math and computer science. Then he went on to Berkeley and got his PhD in computer science. Now he's working at Google in Silicon Valley and really enjoying it.

BK: I have one last question. What advice do you have for young researchers who are interested in starting a career in computer science or mathematics?

SC: Yeah, that's a really hard question. I guess my only comment there, as far as computer science goes, I think they need no encouragement because right now, as you know and I know, undergraduate institutions are being flooded with people who want to take computer science. In our department, really it's even more than we can handle. We have to turn them away. So somehow computer science has become an enormously interesting field, and probably with some good reason. I mean I think the big driver now is artificial intelligence and machine learning. We're all seeing exciting things happening from that. And of course it's a good field to go into, no question, but there's going to be lots of competition.

BK: So I guess… I don't know if… I mean I've asked all my questions, but I don't know if there's anything else you want to add.

SC: Oh, I think you covered it pretty well.

BK: Good. Well, I really enjoyed talking to you.

SC: Oh, really enjoyed your questions, and very nice of you to come. How much time have you taken? I don't know.

BK: I'm not sure. Probably… It's 2:28, so it's probably been about an hour.

SC: Well, I hope they consider that to be enough.

BK: [laughs]

SC: Don't want to bore people too much.

[end of recording]