Oral History of Albert Meyer

Interviewed by:

David C. Brock

Recorded September 5, 2018

Cambridge, MA

CHM Reference number: X8784.2019

Brock: Well, thanks again, Albert, for talking with me. I saw that you were born in 1941, and I wondered if you could tell me a little bit about your family background and your family of origin.

Meyer: That's very interesting because I formally retired almost two years ago now. So I have much more time than usual, and I'm using it for various retirement projects, the latest of which happens to be tracing family genealogy, and I have just accumulated a family tree with six hundred and seventy nodes going back to the 1600s in one line. So, I have learned something or other about my family. But specifically, my grandparents on the side that I understand fully were immigrants from Eastern Europe – Belarus -- I think. And my grandparents were Yiddish speakers, who spoke English haltingly, but at home, the conversation was in Yiddish, which was irritating because I didn't understand any. So, that's what they could talk when I wasn't there. My parents-- my father, who was the child from that group that I'm talking about, understood Yiddish but would not speak it because he was committed to being assimilated. He became a lawyer, and all his brothers became professionals coming from a completely poverty-stricken immigrant family.

Brock: When did they come?

Meyer: They-- I think that my grandmother emigrated as a five-years-old in the late 19th century.

Brock: Okay.

Meyer: As did her husband, although I know much less about that grandfather, who was something of a pariah in the family for good reason. When you get into this genealogy stuff, this stuff sort of turns up--

Brock: Oh yeah, human stories, right?

Meyer: I'm just exchanging email with a newly discovered second cousin, who doesn't show up in any of the family trees. I got her through 23andMe and the genetic link. And it turns out that her father, who is a documented first cousin once removed of mine, never married her mother.

So, there's a whole branch of family that never showed up before, although she's agreed to be integrated into the tree. And so, this was a man with three wives and a mistress by whom he had at least one child in the 1920s.

Brock: Wow.

Meyer: And it's a story that I'm looking forward to going more into. But anyway, coming back to me and my family, the immediate one was then my grandmother who was one of nine siblings, all of whom either immigrated to the U.S. together or were born here. She came as a five-year-old. That group of nine siblings produced about fifty offspring, who then become my first cousins once removed and an army of second cousins that I haven't managed to finish counting yet. This grandmother had four sons of whom my father was the eldest. All of them became-- well, three of whom became professionals, and one of whom became a successful small businessman. And then comes-- but not very many children from those four. Basically, one of them had no children, the second oldest, the doctor. My father had just one child, me. His other brother had just one child, my cousin Barry [Barry M. Meyer], who became a very distinguished businessman, CEO of Warner Bros.

Brock: Oh, my goodness.

Meyer: And the youngest brother was a veterinarian who had two children, one of whom was murdered when he was in his twenties hitchhiking. It was in that era of the '70s when people did stuff like hitchhike.

Brock: Right.

Meyer: And then a daughter who became a professional woman. So, it's an interesting rise from absolutely poverty-stricken immigrants to the next generation a bunch of professionals and successful businessmen, and the third generation, being me, a distinguished professor, and my cousin the CEO of Warner Bros. So, it's definitely a story of rise. I'm not sure that my own children will contribute to the heights, but they're doing fine. And so, that's--

Brock: It's a high bar to exceed.

Meyer: Okay so, that's kind of a little bit of background.

Brock: Was this-- was your family in the Boston area?

Meyer: New York.

Brock: Or the New York-- New York area, okay.

Meyer: The Bronx.

Brock: In the Bronx?

Meyer: Yeah, and a very large contingent of them kept up with each other. I remember, later as a twenty-one-year-old, visiting the last maybe of my grandmother's siblings, her youngest sister Charlotte, whom I had a conversation with about the family. And I asked her what happened to this very large family that was always around my grandmother's house. Her little tiny apartment was always filled with armies of people. And I said, "Well, you know, I don't know. Grandma died, and I never saw them all again. What happened? I mean they all loved each other. It was wonderful having the family around." And this aunt looks at me and said, "Loved each other, are you kidding me? They hated each other. The only thing that kept them together was your grandmother," who was the matriarch. She died, and they turned their backs on each other, never spoke again. Okay so, you know what a child knows and what you find out later.

Brock: Yeah, right.

Meyer: And then this same aunt was the one who told me about my grandfather about whom I had known nothing, and it emerged that he had abandoned-- well, his wife to go live with another woman leaving the wife with four young sons. And he very occasionally contributed child support.

Brock: So, she must have been a very powerful figure to hold everyone together.

Meyer: She was a formidable figure. Her sons really were strongly attached, and I would say intimidated by her actually. Her opinions were tremendously important and influenced them.

Brock: Was learning and education sort of a central kind of a theme and impulse for the family?

Meyer: You know, I don't know.

Brock: Yeah.

Meyer: I would say that there was-- these were people who were not intellectuals at all. I'm not sure that I ever heard an intellectual conversation in the family. And I'm not sure that my grandmother read. She only had an eighth-grade education. But a huge emphasis was placed on being smart. That was what the family was. And I think that-- when I think now, as a seventy-six year old, reminiscing about who I am and how I got to be this way, I realize that it was very important to me to be smart because I also was persuaded, I think inaccurately, but persuaded that I was a particularly difficult and obnoxious child and even continued so as an adult, and that the compensation for that, the thing that would make me acceptable to other people, was how smart I was and what I could do with it. And that I believe-- I've never-- I haven't been through psychoanalysis, although I've had a lot of therapy in my time. But my guess is that that was a theme that added tremendous motivation to me to be doing smart stuff and excelling in my education right through from elementary school on.

Brock: Was this a message that you were getting from your parents that you were difficult to handle?

Meyer: I think it was, yeah. I think it was from both my mother and my grandmother and maybe other-- I got the impression that that was kind of the shared model of Albert in the family and the uncles and cousins that would talk about it.

Brock: What do you think that-- what's your impression of what you were like that they modeled in that way? Were you-- did you have a lot of energy?

Meyer: I think I was needy. I think I was needy. My father was a flawed, deeply flawed man, was an alcoholic, a barbiturate addict, depressive, just-- and basically not there. So, I really had essentially no relationship with him. He was around, but we never talked. He never paid attention to me and so on. And my mother was a very complicated woman that still makes my head spin in

some ways. Everybody loved her because she was always happy, and that was her policy. And she never ever would lay her concerns or anxieties or problems with other people. Those were deeply hidden. And so, she made you crazy. I remember one time telling her, when I think it was in college, and I said that I was unhappy about something. She said, "Oh, you're not unhappy," you know, "You must be-- you're really a happy person." It's like, "What the fuck?" So, that was why everybody loved her. And in fact, I remain disappointed and pissed off at her, although I also loved her and was-- I felt the loss when she died.

Brock: Yeah.

Meyer: But I think that she really made me crazy.

Brock: Well, yeah, because nobody's ever happy all the time. So, how can you--

Meyer: Yeah.

Brock: Learn to deal with what you're-- yeah, I--

Meyer: And her-- I mean her model was denial. It made me-- I think it's another thing-- actually, I don't know whether this is true, but it's a good story about becoming an intellectual because I remember sometime in high school or college, somebody asked me something. And I immediately knew the answer or had an opinion about it. And then I remembered this flash in the back of my mind saying, "Wait a minute, you got that from her." It's probably wrong. It's time to start footnoting all these opinions. Where the fuck did they come from? Do you really believe that, because she was an endless source of misinformation telling me about all of these wonderful people who cared about her and the family who turned out to be completely indifferent then and not such nice people? And everybody was wonderful. Nothing was ever wrong, and on, and on, and on, and then just completely unrealistic. So, her method of dealing with the world was denial. And that's not a terribly effective approach.

Brock: Yeah, a short-term approach.

Meyer: And it continued to her death. I mean a few weeks before she died, she was giving me detailed instructions about arranging for the hairdresser to be in the house so that the moment that she got home she could have her hair done, so she'd be able to see company. And you couldn't have a realistic discussion with her about that. So, that's who she was.

Brock: And so, where were you proving-- where were you proving your smarts? Was that at school?

Meyer: College-- in school.

Brock: In school.

Meyer: I was sort of an extraordinary student early on. I guess now that I think about it in terms of being difficult, I was something of a disciplinary problem in the fourth and fifth grades. And I remember having a conversation with I think the-- maybe it was the junior high-- the elementary school principal because they gave some standardized test in the fifth grade. And I broke it. I mean I got all the vocabulary right and one reading comprehension question wrong, which I actually knew the answer to, but I said it in a way that didn't fit the standards template. And they decided they would transfer me in the sixth grade to some intelligence-gifted program across town on that. And the principal started saying, "Well, how come you never used all these words that you proved you know." It was the usual answer of the "oatmeal's too salty," if you know that joke.

Brock: No, I don't.

Meyer: Oh, I'll come back to it. But the answer is nobody ever asked me before.

Brock: Right.

Meyer: And I think that I was-- school was slow and kind of tedious.

Brock: Right.

Meyer: And I was a brilliant kid. I mean--

Brock: Were you going--?

Meyer: I mean there's documentary evidence of that.

Brock: I will take your word for it. Did you go to the public school--?

Meyer: Yeah, public school system, yeah.

Brock: And did you supplement with-- if it was kind of going too slow, a little boring for you, did you supplement by going to the library and reading?

Meyer: Yeah, I read all the science fiction books in the local library.

Brock: I was just going to ask about science fiction.

Meyer: And like that. I'm not sure that I remember extensive other reading beyond that in mutual sort of assigned books from high school, I think none of whom made a real impression.

Brock: But science fiction did?

Meyer: Science fiction from early on, I think even when I was in elementary school, I was reading science fiction. And, I don't know, I think I was interested in the museum. My mother took me to the Museum of Natural History on many occasions. And I went-- I learned, when I got a little older, to go myself and look at dinosaurs and stuff like that. But school was the thing that I really did very well.

Brock: And did math kind of shine at that early time, or was it across the board?

Meyer: I would say so. I remember in whatever grade somebody was teaching us to do square roots. I've learned how untrustworthy these memories are. But this is a sincere memory that might be true: I have a distinct vision of being in a classroom, maybe the fourth or fifth grade, when some teacher was explaining the mechanics of calculating square roots on a blackboard. And I remember thinking about that and at the time thinking, "Well, yeah, I understand the procedure now, but I would like to be the person who figures out how to do this. Okay, that's

what I want to be." And so, it's one of these things where I wanted to be a sort of a-- essentially, a research mathematician from fourth grade. And I have actually never wavered. I never changed career, never stuck my head up to look around at broader opportunities and interests because that's what I wanted to do, and I stayed with it. And it was tremendously satisfying for decades.

Brock: Did you have a chance to meet some people who-- like, what was the first time that you met someone who was doing that kind of work, so had a real vision of what the life was like?

Meyer: Well, not until really way late, the end of college. I think prior to that, the people that I met that I kind of shared those interests in with were fellow students. I'd like the other smart kids. So, in the sixth grade, they transferred me to this intelligence-gifted class where I was for one-- just one term, the final term of sixth grade. Then I went to Bronx Science [the Bronx High School of Science].

Brock: Okay.

Meyer: And there was an army of fellow students whose focus on this, again, entrance exam school for smart kids was on being smart, being academically accomplished. And I made some lifelong friends there and loved that and was I guess then considered to be one of the guys who's really good at math and starred at it. Although, I had a personal crisis in going between the advanced placement junior year and advanced placement senior year for which you had to do some homework. And I think at that time, I was kind of burned out. I'd gotten like a hundred percent on the geometry Regents exam, which I was capable of teaching that class at the end of it. But I was so over-involved and over intense that I think, when we had to go further into solid geometry, it was, like, I don't want to think about this anymore. I'm fed up. So, the result was that I didn't do that preliminary summer work to be admitted to the advanced math class in senior year of high school. So, I took another non-calculus advanced class, which was fine. And I enjoyed it. Although, there was always the sense of what happened to Albert. How come he didn't join us on the pre-calculus thing?

Brock: Yeah.

Meyer: It didn't do any harm. And to jump ahead, by-- there was an analogous situation that happened to me in my senior year in college where I'd had mono at the end of my junior year and skipped a bunch of exams. But I was also depressed. I think it was hard to separate out the mono symptoms from the depression symptoms. And the result was that in order to-- I had a-- I was excused from about three final exams. And I had to-- I made up one of them, and I didn't-- just didn't make up the other two. I didn't feel like it, didn't want to. To hell with it. So, the result was that I was on academic probation my senior year in college. But I knew I needed some As, so I took a mathematics for non-mathematicians class, which the other kids in the class were incredibly pissed about. But it was fun. I enjoyed it. I actually learned something. And I got the-- one of the needed As that restored my record in senior year.

Brock: What did-- did you have other hobbies or activities outside of this intense kind of work in the classroom? Reading still in high school?

Meyer: I don't know. I think I read. I think that-- as far as I remember, all that intellectual work was focused around high school, high school projects. I think I was on the high school science team maybe, or not the math team that I remember. I did some research projects in social studies and with-- but in class with fellow students. You know, it's a good question. I do not remember doing any extra-intellectual work beyond school.

Brock: Some people who I've talked to who have kind of had a life in mathematics or computing, there's sort of a-- there's a kind of a visual camp, and then maybe there's a-- I don't know what to call the non-visual camp. But some people, their mathematics is a strong kind of spatial visual kind of intuition. And other people, it's more pattern, kind of music, that sort of thing. Which-- I mean if that's at all gets at anything.

Meyer: That makes sense. It's something I've actually thought about because one of the experiences that sticks with me this day was as a freshman at Harvard, they give you this vocational test or interest test. And I remember being advised that I didn't look like the profile of a mathematician, that I looked more like an engineer because mathematicians like music. And I didn't like music. To which I thought, "Fuck you."

Brock: Yeah, probably the perfect response.

Meyer: I didn't say that, but I definitely thought that.

Brock: Yeah.

Meyer: And I would say that I have subsequently learned that there's a whole string of mathematicians. I think of them as the "analyst," the continuous mathematicians, who very typically have a deep connection and interest in music. But there's another kind of people that I've worked with in which I would consider myself more like, the symbolic people, the logical people, the people who do discrete math, symbolic logic, and so on. And they are not particularly musical that I'm aware of. I am really not musical at all.

Brock: Okay.

Meyer: I go to an occasional concert, and it's fine. But it's never something that's much interested me, and I absolutely can't stand having music in the background. I find it a distraction. You know, if somebody's playing the radio in the house, I come and turn it off.

Brock: Well, could you talk about your-- the route that led you to be an undergraduate student at Harvard?

Meyer: Well, yeah it was maybe one of the few times that my father actually interacted with me when, at some point in senior year, he had asked me what my plans about college were. To which I told him that I had already been admitted with scholarships to Cornell and Columbia and maybe then-- of course, City College or something and that I thought I would go to Columbia. To which his response was, "Well, what about Harvard?" To which I said, "Well, I didn't bother to apply to Harvard," because I already had this early scholarship from Columbia. I thought that was good enough. And he said, "Apply at Harvard." So, I did, and I was admitted, again, with an honorary scholarship because he had-- my father had been making enough money then as a lawyer for a large union that we weren't eligible for financial aid. So, I decided to go to Harvard with another seven class-- another six classmates. There were seven of us from my year at Bronx Science.

Brock: From your graduating year?

Meyer: Yeah, class of 1959 that went to Harvard.

Brock: Did you-- did that group stay together once you got to Cambridge or--?

Meyer: Oh, yeah.

Brock: Okay.

Meyer: We were in regular touch with each other all the time. And there was a group, at least two of those people went on to really great heights, I think. One of them is one of the most distinguished geneticists in the world, David Botstein, who's the sort of co-inventor of the genome sequencing project along with Eric [Lander]-- I'm blanking on his name at the moment, the head of the Broad Institute.

Brock: Oh, I don't know him. I'm sorry.

Meyer: Anyway, David's been-- for thirty years, people have talked about his eligibility for a Nobel Prize. He did get one of these super prizes. It's not a Japan prize, but somebody's giving out three million dollar grants to distinguished scientists. He got one of those a few years ago. And the other is a guy named Todd Gitlin, who was--

Brock: Oh, I've heard of Todd Gitlin.

Meyer: Well, Todd is a sort of a very prominent left-wing intellectual author of multiple books.

Brock: NYU or something?

Meyer: I think he's at NYU now. He had been at Columbia. He's a regular contributor to the op ed of the *New York Times*, and the *New York Review of Books* and so on, very prominent sort of left-wing intellectual. So, Todd was the first in our class in high school. I was second. But there was always a sense that Todd really was head and shoulders above everybody.

Brock: And he was part of the group that came to--

Meyer: He was part of the group that came to Harvard and other-- and he, again, he was one of the people who represented a kind of-- who was daunting to me in that in his sophomore year, he decided to give up mathematics because it was too easy and not important enough, and to shift over into politics and policy. Whereas-- and Todd, of course, had always been I thought better than me at math. In freshman year, he was grading third year calculus.

Brock: Wow.

Meyer: And here I was, the thing that I wanted to do and aspired to, he was sort of saying well, it's too easy. He's moving on to better things. But they were better for him, I guess. He then, I remember him saying the one thing that he would do is he'd take the Putnam exam again because that was an easy way to pick up five hundred dollars. He expected to win. I think he came in second or something. He had that kind of mind. It was very impressive.

Brock: So, did you-- when you got to Harvard as an undergraduate, did you gravitate toward mathematics?

Meyer: Oh, no, I was a math major. I wanted to do that from the beginning. But it was another personal problem. I didn't do well in math classes. And I had-- as would be inevitable for somebody at that age and state, you wonder whether it's you or them. And I suspected it might be me, but I was sure it was them.

Meyer: That it was the Harvard maths department at that time, there was no concept of teaching. There was the-- what they wanted to do for creating a program for gifted kids who got super high scores on the SATs, which I was in. So, they put their young genius in charge of it, a guy named Shlomo Sternberg, who was then twenty-five, very famous genius mathematician. So, he didn't have a clue about teaching. I remember the first lecture he gave. He stood in front of the class and said, "Okay, a vector space is an additive group with a multiplicative ring attached and connected by distributive law. And they're called vector spaces." He writes on the board, V. That was enough for me. I dropped that class and felt, "Okay, whatever's going on here, this is not for me." And it wasn't until-- but I was looking for stuff to do. So, it wasn't until my I think junior year that I took a course called applied math. I didn't know what was exactly. But they were second class citizens. They were in the Harvard School of Engineering, a division of

Engineering and Applied Science, not on the mainstream liberal arts school and not in the math department. So, they were much more defensive. And they actually felt the need to do what the mathematicians didn't do, which was they needed to justify what they were doing by explaining the point to you.

Meyer: And I found that as soon as I understood the point, I could do it. I needed to know why are we doing this, what are we aiming for, what are we trying to accomplish. And once I had that, then I could run with it.

Brock: What was that view of the intent and intention of applied mathematics that they were presenting?

Meyer: Okay, well this is the big idea and brings us back to Dennis [Ritchie] because this was a time when the-- well, first of all, there were hardly even any computer science departments in the country then. This was in the early '60s. And there was just coming to be a vision that there was something special about computation. In the 1930s and '40s, the notion of what was and wasn't computable was very extensively worked on, was understood. There were logical limits due to Gödel and Turing about what could be computed and what couldn't be computed. But the new idea was that, "Let's try to understand what you can do with computation." And in order to understand what you can do with computation, that was when the idea of computational complexity came into being that yeah, there were things you could do, all sorts of things you could do with computation, but not all of it was easy. And you could actually try to get a theory of which things were theoretically doable but were so impossibly hard that, pragmatically, you couldn't do them. So, there was a sudden new vision that was an extension of these ideas in the '30s of "yes or no" computing where in the '60s, it began to be a degree of computability. How well could it be computed? Which led a decade later to the original work on the P = NP question, which is now one of the Clay questions [Millennium Prize Problems] and is considered the central question of theoretical computer science, which was formulated by a fellow graduate student of Dennis and mine, Steve Cook, who was a year ahead of us in graduate school and who we both interacted with.

Brock: How did you-- can I-- would you mind indulging me? I would love to hear how you explain that P=NP question.

Meyer: Okay.

Brock: For somebody who's not an insider.

Meyer: Yeah, okay. I can try. I'll-- let's begin with P, which is the easier thing.

Brock: Yes.

Meyer: So, you want to have some notion of computations which are hopeless. And sort of the definition of hopeless generally is if the effort of computing an answer grows exponentially with the size of the question.

Brock: Yeah.

Meyer: That's-- you're going to wash out very quickly. It means that every time you increase the size of the question by one bit, you increase the difficulty of computing the answer by a constant factor that's greater than one. Say, for the definition, it's two. So, I make-- so, I go from the question of a ten-bit question to an eleven-bit question. The eleven-bit question takes twice as long to answer as the ten-bit. Twelve-bit is four times as long as the ten-bit and so on. That's exponential growth. And when that hits you, basically, you get stymied at very small numbers. And you have to throw your hands up and--

Brock: Got it.

Meyer: You get to beyond astronomical numbers by the time you get to a few hundred digits, which is the basis of modern crypto: That you have passwords of a few hundred digits long. It's impossible to crack them because the only way to crack them is by an exhaustive search that exhausts all the resources of the known universe.

Brock: Yes.

Meyer: Okay. So, that's what we can't do. If it's exponential, it's off the page, at least conceptually. I mean those are broad strokes. And there's various constant factors and things that can impinge when this asymptotic growth becomes relevant and practical. But let's just keep it simple and say if it grows exponentially, it's off the table. Well, the other standard thing that doesn't grow exponentially are polynomials, things like n^2, n^3, n^4. So, as a broad stroke notion of what's not exponential is polynomial. Okay, so very roughly to the first-degree discussion, polynomial means potentially feasible. The truth is, if it's a hundredth-degree polynomial, it's just as off the table as an exponential, but nevertheless, the difference between polynomial and exponential is a very dramatic one. And it's a great breakthrough when you can show that some exponential problem is actually, you can suddenly find a clever way to do it in polynomial, even if it's a high degree polynomial.

Brock: Okay.

Meyer: The optimization-- the optimistic approach-- assumption is if it's a high degree polynomial, that's a first pass. Probably, you can knock it down. So, for example, prime detection I think was originally an eleventh-degree polynomial. It's down to about three now—that is detecting primality, recognizing whether or not an integer is a prime.

Brock: Okay.

Meyer: So, the simple thing is exponential versus polynomial. Now, comes the weird idea of problems where we may not know how to find the answer easily, but if you give me the answer, there's a way to check it in polynomial time.

Brock: Okay.

Meyer: I'm trying to think of a simple-minded example. I give you a whole bunch of boxes that I want to pack them some optimal way into some container. If I want to show that some particular proposed way is not optimal, all I have to do is show you a better way to do it. And it's very easy. If you give me the better way to do it, it's very easy to check that. Okay? One other hand, if I have to find the better way to do it, I'm floundering around through an exponential search, typically.

Brock: Okay.

Meyer: Okay. So, that's an example of an NP complete problem. That is one where we don't know how to find the answer in polynomial time. The obvious way to find the answer to any given question is one that requires a search that's exponential in the size of the question or appears to require that. But it has the property that if you give me that answer, I have a way to check it very fast.

Brock: I have a way to check that it is indeed a good solution?

Meyer: Yes.

Brock: Okay.

Meyer: So, again, let's go back to the business of packing boxes. We can-- simpler than boxes. Just pack a bunch of different size intervals, integer intervals, and pack them into a container of length-- on the line of a given length. Like I've got something that's a hundred inches long. And I give a bunch of pieces of various eleven inches, seventeen inches, four inches, and so on. And I want you to get the boxes full as you can manage.

Brock: Right.

Meyer: Okay?

Brock: Mm-hmm.

Meyer: And so, you come up with one. And you say, well, I can't fill up the whole hundred inches, but I can fill up ninety-eight of them with the pieces of sizes you've given me. And I claim that's the best you can do. Well, it's easy for me to show that that's not the best if it's not because all I do is show you one that does ninety-nine or a hundred. So, I can instantly check whether this new packing is better than a given packing.

Brock: Right.

Meyer: Okay. On the other hand, if you say okay, I did ninety-eight. Let me see you do better. Searching to determine whether the ninety-eight really is the best you can do is something we don't know how to do. Seems to be exponential, okay. That's the characterization of the NP-complete problem -- the Nondeterministic Polynomial time-complete problems -- never mind where the nondeterministic comes from. The efficiently checkable problems, okay? Efficiently in the sense of you can check the answer in polynomial time if, somebody gives it to you, but if you not given it, then we don't know how to find it, and the conjecture is that you can't find it in less than exponential time. The conjecture is that the NP-complete problems, these checkable problems, do require exponential time to solve, but that's the open problem. It's been open for, what, 50 years?

Brock: To show that that is-- to create a proof.

Meyer: Well, one way or the other. I'm going to find a way to knock them off in polynomial time. Or prove that you can't.

Brock: Find an example, right? Where you can check it and you can solve it in polynomial time.

Meyer: Well, no. Finding an example is not the difficulty, because one of the insights is that the problems come in hundreds of different guises, but it's really, in a certain sense, the same problem.

Brock: Oh. Yeah.

Meyer: So we know exactly what examples of the-- like the packing problem I gave you is an example of a problem. It's enough to focus on that. If you can do that in polynomial time you can do them all in polynomial time. If you could prove that that one requires exponential time, they all require exponential time. End of story.

Brock: Okay, okay.

Meyer: Okay. So these are these peculiar things that are right on this funny boundary between polynomial and exponential that can be done in exponential time but we-- and we wonder

whether you can do better, but they have this peculiar property that you can check the answer fast, efficiently, and nobody really understands those.

Brock: Am I wrong to perceive, like, when I was reading Dennis Ritchie's thesis and when I was reading your paper with him of '67 or 9?

Meyer: '67.

Brock: '67. "The Complexity of Loop Programs."

Meyer: Right.

Brock: There's very strong echoes of this kind of concern or pre-figuring, what we just talked about of the P=NP, about the complexity of programs and--

Meyer: No, actually, so P=NP comes another half a dozen years later, but prior to that we were very focused on this issue of "How do you recognize problems that can be solved by computation but there's no efficient way to do it?" Okay. And that was, those papers by Dennis and me, was exactly about that. We were showing that structurally, in terms of syntax, they were problems that could be solved by programs with a complicated nested loop syntax that couldn't be solved by programs with less nested syntax. So we were kind of trying to explain the power of the programming description syntactically and say, "Yeah, there's a payoff for having a messier program. You can do more."

Meyer: Okay? Okay. That's intuitive, but now it becomes a theorem, and that's pretty good math, when you can actually prove that. Now, the insight that comes from both Dennis's work and my work and our joint work together was that in a certain sense, the trick to proving it and understanding what's going on is to get away from the syntax and realize that it's simply talking about how long the computations are allowed to run: That the programs that have depth of nesting of loops to *n* are the ones that run in a certain amount of time that is the *n*th function in a list. If they run in less time than that within that function-- these are, by the way, functions that grow much more rapidly than exponential but, you know, we weren't focused on exponential then. If you can do it in time, in this amount of time, then it's got a simple syntax description,

and if it takes more time than that it doesn't have a simple syntax description. So what we were really doing, the key theorem to understand the significance of the syntax, was that having more time to compute you could do more, and that was a theorem that it was easy to prove using the methods from the 1930s of diagonal arguments and Turing arguments. More time is better than less, and then we connected these syntactic questions about the power of syntax by saying, "Look. Don't think of it as syntax. Realize that this level of nesting syntax just means it's computed on a certain amount of time, and if you give me deeper nesting, I can run for longer and I can do more." End of story.

Brock: When you were-- so when you were getting into applied math in the early '60s and computability and these kind of questions were-- they were at the heart-- well, they were being expressed as, you know, central to applied mathematics?

Meyer: Well, no. That's much too broad. This was a small group of people that were working on this. Applied mathematics was a huge subject in which this kind of theory of computation was a tiny, new part.

Brock: I get it. A subfield.

Meyer: A subfield, okay.

Brock: Okay.

Meyer: That it was a very small community of researchers, but I happened to connect up with a guy who was doing that, Patrick Fischer, who was my actual supervisor, as he was also Dennis's actual supervisor, although I think officially he left Harvard before either of us got our Ph.D. So he's not listed as our actual supervisor by Harvard.

Brock: Okay. I wondered about that. Yeah.

Meyer: Yeah. Yeah. But Patrick, our real supervisor: he supported us, he suggested the research and so on. So he was our actual thesis supervisor. Patrick was very much interested in this notion of understanding the nature of computation, what made things hard, what made things

easy, and they were approached in various ways. One of them was, by syntactic classifications, "What kinds of things could different kinds of programs do?" Was related to that. Was extensive effort at the time in automata theory, where you had these various restricted models of things that were weaker than Turing machines, like pushdown machines and finite state machines and so on, and you sort of ask, "Well, how do we understand what this more limited model of computation can and can't do?"

Brock: Oh, okay.

Meyer: Okay. So that was-- the idea was, we were trying to understand that to really understand computation. You needed to understand what you couldn't do with something. You didn't really understand the power of something until you understood the limits of it, and so that was a top-level theme which I learned early, and I understood that point of that, and that was what underlay a lot of the work that we were all doing at that time.

Brock: Okay. So when did you-- so you encountered, I think, in your junior year applied mathematics. When does-- does that also involve starting to have direct experiences with computers?

Meyer: Yeah. So that was-- it was in junior year. I took a course called *Introduction to Applied Math* from a guy named Tony Oettinger who was a leading researcher in machine translation, of syntactic translation, at the time, and I, you know, it was a very easy course for me. Not a problem. But you got the sense that there was a point to it, that it had to do with this understanding computing and how it worked, and then that led me to taking, I think, in my senior year a couple of graduate seminars, one of which was taught by a young teaching assistant named Shimon Even, who went on to become a very prominent theoretical computer scientist, and another one I think was taught by Pat Fischer, which was an introduction to recursion theory. That might've been first-year graduate school. I'm not quite sure.

Brock: Okay. Yeah.

Meyer: But that was the first time that I was actually meeting and learning from and working with in an intimate seminar kind of setting with people who were really doing math and doing

research, and it was, you know, tremendously exciting and engaging, and it was what I wanted to do.

Brock: Was Dennis Ritchie an undergraduate with you?

Meyer: You know, I--

Brock: I think he--

Meyer: He must've been.

Brock: Yeah.

Meyer: I don't remember seeing him at any reunions or anything like that, so we had nothing to do with each other, I think, as undergraduate classmates. We must've met at-- it must've been the summer between first year of graduate school and second year of graduate school that Patrick Fischer had hired both Dennis and me as research assistants, and this is the story I was going to write up for you.

Brock: Yes.

Meyer: And he assigned us to work on the same problem and never told us.

Brock: Oh, my God.

Meyer: Not only that, I remember how exciting it was that I would finally get to, you know, I'd taken his class. I got hired because I took his class and I think I got a hundred percent on the final exam and so on, and it was another one of those classes that by the time that class was over I could teach it. I knew it. I had, you know--

Brock: You really got it.

Meyer: --I was really into it, and again, I sort of understood the point of it. You were going to figure out what you could and what you couldn't do and why and the structure of all of that. So

Patrick offered Dennis and me these research assistantships for the summer. Dennis might have been in that class, but I don't remember having any interaction with him other than I knew that he was another student of Pat's and somebody that I thought would be interesting to kind of talk to. But what happened was that, you know, Patrick says he's going to hire me for the summer and there's this list of papers-- I actually found the paper-- there's this list of 11 open problems, he points to number four and says, "I want you to work on that one." Okay. So, "Fine." So I said, "Okay." Well, you know, I don't know what the hell it was, but, "You want me to work on that, that's fine." So I'm happy. "I'll read the paper right away. When do you want to meet?" And he looks at me and says, "Meet? I'm going to be away for the summer. Just work on it."

Meyer: So I remember that as being one of the more miserable times in my life where I spent the entire summer, and I'd sit at my desk for hours at a time reading this paper and rereading and thinking about it. Have nothing to show for it. I could never tell whether I was actually working or not.

Brock: Right.

Meyer: And it made you little crazy, and it was a kind of a, a first test of could I do research or not, and I failed it. At the end of the summer I felt I had nothing to show for it. I told Pat that I had this one result that I felt was very minor and he was-- he was a very encouraging guy. He said, "Oh, well, that's fine. Write it up." I never did, because in fact I don't believe it was worth writing up, but it was nice of him to say that. But I remember telling him when-- at the end of the summer when we got back together, that I said, "Well, listen. You know, I had decided that I would give one year to graduate school, see whether in fact I really had talent for the subject, so see whether I could get anywhere, and I worked, you know, I thought I worked really hard for the summer. I just couldn't get anywhere. Had nothing to show for it, so I think I'm going to quit, try to find some other career and thing that I'm good at." And Pat was kind of disappointed but, you know, what are you going to do? But this was at a time when-- it was just the beginning of the term and since I had no other plans it wasn't as though I was going to drop out, had to drop out then.

Brock: Right.

Meyer: So again, this is a story that I believe is true, but I've told it so many times that who knows the extent to which memory has embellished it. But after this conversation of saying, "Well, I've decided to quit graduate school, because I didn't get anywhere and I don't see any evidence that I have, that I'm going to excel in this field." We're walking down the hall towards his graduate seminar, and I remember thinking, "Okay. If I had to give a presentation at the graduate seminar explaining what the issues were, what he had assigned me to work on, what I was trying to do and how to do it, what would I say?" and I'm sort of-- as we're walking down the hall I'm mentally thinking this through and I say, "So you put these pieces together and here's how you solve it." I'm like, "Fucking solve this?!"

Brock: Walking down the hallway.

Meyer: Walking down the hall. I swear this is my sincere memory, whether true or not.

Brock: Yeah.

Meyer: It's so many years later, but I do remember kind of excitedly at the end of class, or maybe even before class, telling Patrick, "I solve it, I solved it, I solved it," and he started saying, well, you know, he's got to give a-- he's got to give a lecture now. Talk to him later, and I remember afterward, you know, buttonholing him and trying to show him how to do it, that I figured out how to do it, and he, again, well, he was kind of a encouragement, that guy, that way, as a supervisor. Whether he understood it or not, I don't know, but it was correct and sound. So that was the story of me and this research and getting-- and so then that was my first result. I'd solved the problem. Was an open problem that my advisor had said was important, so it proved that I could go somewhere. So then a little while later, I was, I would say, surprised and a little disappointed to hear that Pat said that Dennis had also solved the problem, which I--

Brock: Hm. Like weeks later or...

Meyer: I don't know exactly the timing, but Dennis was working on it over the summer, and I believe it. I believed it at the time, because first of all, it wasn't that important to me whether or not somebody else had done it. What was important was that I had done it. It was real. But the other thing was that I learned either from Dennis or from Patrick, I don't remember clearly, this

concept of loop programs, which was Dennis's invention, and it was so beautiful and so important and such a terrific expository mechanism as well as an intellectual one to clarify what the subject was about, that I didn't care whether he solved the problem. I was delighted to have him as a co-author because loop programs alone were worth it. Okay. Loop programs made into a very simple model that any computer scientist could understand instantly, something that the traditional formulation as, you know, in terms of primitive recursive hierarchies and Grzegorczyk hierarchy with very elaborate logician's notation for complicated syntax and so on that would make anybody's eyes glaze over. Suddenly you had a three-line, four-line computer science description of loop programs. So that's-- never mind what these other formulations are. That's what primitive recursive functions are, and all we're interested in is how much you can do when you nest the loops to depth, a given fixed depth, and so on, so... So I believed, actually--

Brock: So Dennis saw the identity between--

Meyer: Dennis saw the identity--

Brock: --that a loop program was a recursive function.

Meyer: That loop programs were exactly a way to formulate the primitive recursive functions, and the fact that they were a way to formulate-- well, what we proved was that they were exactly a way to formulate the so-called Grzegorczyk hierarchy, which was a classification of the primitive recursive functions.

Brock: Okay.

Meyer: By syntactic categories in terms of, again, the complexity of their definitions in terms of this weird logic formalism and which were proved to be a proper hierarchy in the sense that the next one, the *n*th one, definitely was larger than the *n*-1st, than the one that came before.

Brock: Okay.

Meyer: Okay?

Brock: So let me just try--

Meyer: Yeah.

Brock: --and replay it to you in a kind of, as I'm, to make sure I'm getting it, is that there is this description of a set of mathematical functions, these primitive recursive functions, and there's a particular ordering of them, and that that is identical, that is the same thing, as these nested loops. You know, that kind of description in terms of the depth of--

Meyer: Yeah, it's--

Brock: --loops--

Meyer: Yes.

Brock: --is the same as this hierarchy.

Meyer: Yes, yeah.

Brock: Okay.

Meyer: I mean, the thing is that we're talking about a class of mathematical functions, functions on the non-negative integers to the non-negative integers. But the way that they're defined is by the structure of the definition that you give such a function. Okay. So we have, it's a kind of a historical irony that Gödel originally formulated the concept of the primitive recursive functions, because he wanted what he considered to be some very simple class of functions that no one could deny that these were mechanically computable. He wanted the simplest possible definition that would be self-evident that, "Oh, yeah. If it's that kind of thing, I definitely know how to compute it."

Brock: Like an increment by one. This is this sort of thing?

Meyer: Oh, well, it was by-- basically it was functions that were defined incrementally that you define the n+1st thing in terms of the nth thing and some operations on it that you know how to do. That's the essence of--

Brock: That's a recursive function.

Meyer: That's the essence of primitive recursion. Okay. And so Gödel wanted those because he needed some class of functions that nobody would doubt that they were computable, and then he could use those as a basis to come up with a much broader general definition of computability in general, which was something that at the time it was considered to be-- needed justification because it was a sweeping generalization, calling all of these functions are exactly the computable functions. Before you can get to there you need some core of functions that nobody has any doubts about.

Brock: That nobody's going to argue with.

Meyer: So Gödel's motivation was he needed this so-called very simple set of functions. But from our new perspective in terms of computational complexity, was to realize this set of functions may obviously be computable, but they grow at a rate that is beyond astronomical description, even, like, at the second or third level. So although logically it's very easy to see that they are very computable, they are in fact so impossibly complicated that you wouldn't even try computing the value of a function value at 5, say, because it would be a number that was-- couldn't be usefully described in any other simpler way. If you actually tried to compute it, the number of digits would -- it's one of these--

Meyer: Fill the universe with atoms kind of thing.

Brock: So the-- because I was wondering when I was reading, both-- well, I was wondering in my reading, you know, "Why the attention on recursive functions?" and it's precisely because of their relationship to computability or algorithmic sort of solving or something like that; is that correct?

Meyer: Yeah, I think so. Look, there's the central question that we begin with here in theory of computation that goes back to the '30s, the work of Gödel and Turing, is "What's computable and what's not?" Okay, and in order to answer, "What's not?" you need a definition. What does it mean for a function to be computable? So what we're talking about are functions for definiteness from non-negative integers to non-negative integers. You know, like, addition.

Well, a pair of non-negative integers to a non-negative integer. You can add them, you can multiple them, you can take one to the power of the other, you can do all kinds of stuff. Okay. You could map it to 0 or 1 depending on whether it's odd or even. Okay. So there's all sorts of mathematical functions and arbitrarily definable functions. How can we come up with a persuasive general definition of which ones can be computed by a program and which can't? If you ignore issues of efficiency. You sort of say, "Is there a program where I put in the number *n* and I get out *f*(*n*), eventually?"

Brock: Right. Right.

Meyer: You know, eventually.

Brock: Right, right.

Meyer: And so the puzzle in the '30s that Turing and Gödel faced was, "What is that definition? What is the definition of the computable functions? How can we define them?" And there were many proposals. I mean, Gödel had one, I think Alonzo Church had another in terms of Lambda Calculus. I forget whether Post was one of the contributors, but--

Brock: I think so.

Meyer: --there was a system called Post, Post rewriting rules. So rewriting systems, and then Turing came along with his description of a very simple computer-like abstraction that you could see, understand, how it works step by step, and he made the argument that was very persuasive, and I've heard anecdotally said that it was Gödel who said, "Yes, Turing is what made it convincing." That this was the general definition of what was computable. Okay. If it could be computed, then it could be computed by a Turing machine, and certainly if it could be computed by a Turing machine, that was a very convincing and persuasive definition that it really was computable in any reasonable, intuitive sense about what a digital computer could do.

Brock: Got it.

Meyer: So that was sort of this big idea, and the-- and again, it's a thesis. It's not a theorem. It's a definition that you're trying to justify. We're trying to define what are the computable functions? I mean, you haven't defined them yet, so you can't say whether it's right or wrong.

Brock: Right.

Meyer: And it's historical evidence, that the people who were thinking about this thing, there were four or five major proposals for mechanisms that could be considered to be mechanisms of general computation, that looked different. One was in terms of Lambda Calculus and, you know, formula-- calculating with formulas by substitution. Another one was by rewriting patterns of symbols. Another one was by recursive definition. Another one was by this atomic Turing machine, computing step-by-step model, and then a wonderful theorem was they all compute exactly the same class of functions. Given one description in terms of rewriting rules, you can find a Turing machine that'll do it.

Brock: That does--

Meyer: Give me a Turing machine and I can find rewriting rules that will do the same thing as the Turing machine does. So that was one of these great, wonderful ideas. All of these systems could be shown to be offering different definitions of the same class of computable functions that is now universally accepted as, "Those are the computable functions."

Brock: Okay.

Meyer: And that was what we were building on, but the insight in the '60s was, "It's time to refine that," because what is and isn't computable now, within the computable functions, there is structure that matters because we need to know how computable it is, and it became a way to reprise all those ideas in the '30s.

Brock: Oh, oh. Yeah, yeah. I see.

Meyer: In the context of, "How computable is it?"

Brock: Right.

Meyer: And again, many definitions of what that should mean, of how computable it should be. It could mean computable by certain kinds of restricted automaton computational devices, like pushdown automata or-- I forget. There were various kinds of pushdown machines, and finite automata and so on, and counter machines was the thing that I worked on extensively then with Pat Fischer and other collaborators. But the other approach was about just asking how long it took the computer, how much space it took the computer.

Brock: Right.

Meyer: And so the problem that Pat had picked out was this business about creating within this class of functions that Gödel thought was very simple and straightforward and obviously computable.

Brock: The primitive recursive functions.

Meyer: The primitive recursive functions. The challenge was, well, this guy Grzegorczyk -- Polish mathematician, logician -- had come up with a way to classify them.

Brock: The primitives?

Meyer: The primitive recursive functions, and broken up into an infinite hierarchy of successive classes of zero, the first and second, all the way up, such that each class was able to do more than the previous one. So it provided this kind of hierarchy within the primitive recursive functions that you could sort of say, "Okay. That's got something to do with how primitive recursive they are, how complicated they are."

Brock: Right.

Meyer: But it was a syntactic definition, and so in a certain sense, you know, the *n*th class could do more than the previous class. Each class could do more than the previous class. But what more could it do? I mean, how would you explain that?

Brock: Yeah, yeah.

Meyer: And in a certain sense, what-- the particular open problem that Patrick sent Dennis and me to work on was to try to explain Grzegorczyk's definition, which, in a certain sense, has an ad hoc element to it, whether it could be explained simply in terms of the depth of nesting of these recursive definitions, of primitive recursive definitions, which-- so that was the question that Pat was setting. Does the Grzegorczyk hierarchy coincide with the depth of nesting of primitive recursions? And Dennis's wonderful insight was he got rid of all his horrible syntax and cumbersome technical definitions of the closure properties of these functions and even the definition of what primitive recursive was, and replaced them with loop programs. They're a gem. Yeah.

Brock: Which seemed to me just like just a very beautifully simple computer program.

Meyer: Exactly. Exactly.

Brock: You know, and I wondered about, when reading his work and your work together, what I wondered about was these loop programs itself. You know, like, there's this idea a Turing machine can compute anything that can compute. Can any computer-- how generalizable is the loop program? You know, that's what I didn't understand.

Meyer: Okay. Okay, fine. The loop programs have a property which guarantees that they can't do everything.

Brock: Okay.

Meyer: Which is that loop programs, from the very way they're defined, they always eventually stop. You run a loop program on a given set of initial values, you know it's going to stop. That was the property that Gödel wanted with the primitive recursive functions. He wanted to be evident-- you definitely could compute these. There was no problem about getting into infinite loops or any other difficulty. Primitive recursive functions were evidently computable.

Brock: Okay, okay.

Meyer: Loop programs just capture it in a more explicit, computer science way. Now, the fact that they always stop is an interesting little induction proof. It's not completely obvious from the way they're defined, but it's a very elementary proof from the way they're defined that they're guaranteed to always stop.

Brock: Okay. Okay.

Meyer: And so that says that they automatically can't do everything.

Brock: Okay, got it.

Meyer: But then it becomes a puzzle of, "Well, okay. They can't do everything. They do some hunk of things." But within the hunk of things that they do, can you-- Is there structure there? Can you clarify the degree of computability by these loop programs?

Brock: Right, right.

Meyer: And it turned out that there was, and that this syntactic depth of nesting was a way to explain the Grzegorczyk hierarchy, which was the open problem that Pat had assigned us. But more than that, the real insight was that you solved the syntactic problem by realizing it and reformulating it as a question about what you can compute in a certain time bound.

Brock: Yeah.

Meyer: Okay. That when you say, "Okay. I have a certain function that I've defined," in a nice, simple, recursive way, but it's a particular definition of a function that grows rapidly, and what I'm asking is, "What can you do by computations that never take more time than that function says they should take?" and so the insight was that all of these syntactic questions about loop programs and primitive recursive functions and so on were really computational complexity questions. They were asking questions about how much you could compute with a given time bound, and you give me more time, I can do more.

Brock: Right.

Meyer: Yeah.

Brock: Right.

Meyer: Which was, again, an easy result based on the 1930s diagonal argument approaches, and so in a certain sense it was a problem we'd been assigned to solve in which we wrote up-- we won a Best Paper Award at an A.C.M. meeting for this paper, which none of the people who gave the award understood, but so be it.

Meyer: They didn't understand the significance of it. They thought it was important. It wasn't. Was a highly theoretical result that had no relevance to practical computation, but they got excited about it and they were happy to have some young theoretician graduate students giving a paper at an A.C.M. conference, so they gave us this award. But the insight was that we disposed of the syntax, stopped worrying about it, because it was really about time bounds and complexity.

Brock: Hm. It seemed that-- I have a lot of questions, but one that has been burning in my mind is about it seemed just like-- well, my understanding of part of what, like, Turing and Gödel were up to were to use compute-- to translate questions about provability or completeness of mathematics, foundations of mathematics, to turn that into questions about computability and learn something about. Use computability to find out things about the foundations of mathematics or mathematical results or about mathematical--

Meyer: I think that's fair, yeah.

Brock: --objects.

Meyer: Yeah, I think that's fair.

Brock: And it seemed that, to me, when I was trying to understand Dennis Ritchie's thesis, it was also, it seemed, he was doing a similar thing there, where he's like, "I can create loop programs and talk about their structure and the depth of loops, and that tell us about practical computability and the time," but that then I'm-- and I didn't fully understand it.

Meyer: Got it.

Brock: But the structure of it seemed to me, and like once I translate these questions about mathematical objects into these questions about computability and loop programs, then what I'm learning about is stuff about the recursive functions, and I don't know if that's correct or not but that's...

Meyer: I need to think about that for a second, so...

Brock: Yeah.

Meyer: Well, look. There's this, there are these classes of functions that are definitely mathematical objects, and-- but there's, you know, endless classes of functions. There's polynomials, there's trigonometric functions, there's functions definable by infinite series and so on.

Brock: Right.

Meyer: So what we were looking at was, what turned out to be the fundamental concept, was classes of functions that could be computed within a given-- with a given amount of time and space resources. Where the amount of resource was defined by a function that was computed from the size of the input. You tell me I'm working on inputs of size a hundred, you can have a hundred times this amount of time steps to do it or a constant to the power of one hundred steps to do it or whatever, but as a function of the size of the input, that's how much time I'm going to get to do it and I wonder can I do it in that much time or not?

Brock: Okay.

Meyer: And so the fundamental question was, "As you tell me about the amount of resource that you're willing to allocate to the computation, what can I say about what you can accomplish with that or not?"

Brock: Ah. Okay.

Meyer: Mm-hm, and the simple insight, it's very intuitive, but-- and it was also easily proved using the methods of diagonalization from the 1930s, was, "Give me more time, I can do more." Okay.

Meyer: Okay. Duh.

Brock: Right. Right.

Meyer: But then, it turned out that that notion of, "Give me more time, I can do more," connected up very nicely with these, with this other approach to try to classify the difficulty of computing functions that was based on the complexity of how they were defined as opposed to how they behaved or how they ran.

Brock: Right, right.

Meyer: And the insight was, "Hey, the two really ought to connect up." That I can tell you from the structure and the complexity of its definition how much computational resource it's going to need, and conversely, this is the mindblower, if you tell me that you can compute it using a certain bounded amount of resource, I can guarantee you it has this syntactically simple description.

Brock: Ah.

Meyer: Okay. So it meant that if you were interested in the syntax, you could understand it in terms of how much time you were spending computing things, but more than that, it told you if you were fundamentally interested in what made computations resource-demanding, you didn't have to think about the syntax. Because it wasn't relevant.

Brock: Okay, okay.

Meyer: The syntax was just another way of talking about the basic questions you were interested in, which is, "How much time and space does it take to accomplish this task?"

Brock: Okay.

Meyer: So that's the key insight in Dennis's thesis, and in my, that early first work of mine, and the stuff that we published together.

Brock: Right.

Meyer: So coming back to Dennis and part of the anecdote was I have never read Dennis's thesis. I never checked on it, because I didn't care. As I told you, I believed that he probably got it, but it didn't matter, you know, even if it turned out he didn't quite get it. It was fine. The Loop programs were enough. I was delighted to have him as a co-author, okay. But in response to the correspondence with you, I went and looked at his thesis, which I had on my shelf for 40 years, <claps> and I actually said, "Well, I should look at it," and sure-- and I still-- to tell you truth, I haven't read it. But I read the 20- or 30-page introduction.

Brock: Yes, the summary.

Meyer: And it was clear that he did indeed. He figured it out his way. It was the same idea as mine, but he had figured it out independently. Okay.

Brock: So this was the solution to--

Meyer: -- to the problem that we had both been assigned--

<overlapping conversation>

Brock: -- the summer problem.

Meyer: -- over the summer. Yeah.

Brock: Oh, my gosh! What a trip!

Meyer: Yeah. So that was the crazy story. I mean, what kind of a supervisor? You know, I feel great affection for Pat, really. He led me-- he made this career possible for me. He supported us, he was encouraging, he defined the problems we were working on. He was a tremendous

influence, both intellectually and professionally. But what a way to manage two students, you know? But it's so wonderful to have someone to talk to about this.

Brock: I know!

Meyer: It was such a painful, lonely summer! And somehow or other, it didn't occur to him to link Dennis and me. He was off doing whatever he was doing for the summer.

Brock: Well, could you tell me a little bit about how-- once you got to know Dennis Ritchie, and you have this very early, important collaboration, what was that like? Yeah!

Meyer: I mean, what's poignant about it, I never got to know Dennis Ritchie.

Brock: Huh!

Meyer: Okay? Dennis and I really had just a-- I liked him. He was a very sweet, easy-going, unpretentious guy. Clearly very smart, but also kind of taciturn. I'm kind of a voluble, you know, conversational person.

Brock: Yeah, right.

Meyer: And I love talking about things, and arguing about them, and stuff like that. And that was not Dennis' style, at all. So we talked a little, and we talked about this paper that we wrote together, which I wrote, I believe. I don't think he wrote it at all, but he read it. The first one he read and made comments on, and so on, and he explained loop programs to me. But we never actually collaborated. It was a disappointment. I would have loved to collaborate with him, because he seemed like a smart, nice guy who'd be fun to work with, but yeah, you know, he was already doing other things. He was staying up all night playing Spacewar in the basement of MIT.

Brock: Oh, okay.

Meyer: And it wasn't-- was it called Spacewar, or it was some kind of space thing with early-- very early computer game of a--

Brock: Shooting ships?

Meyer: Well, the main thing was that they were in the gravitational pull of a strong sun. I mean, you can navigate in trajectory, so that you didn't fall into the sun, but you could use it to make amazing curves, getting up behind your opponent. So apparently Dennis was all caught up in that. That summer apparently that we were both working on this problem, he was spending many hours at night hacking this computer game. And so, in fact, I would say that although we collaborated on that paper, we never really worked together. And there was a second paper that we-- he was a co-author with me on which was stuff of mine that was related to what we were doing, and I felt that he should be a co-author, because it grew right out of what we were doing. And, but, and I think I remember trying to get his attention, and maybe even his formal permission to put his name, and I never heard from him. He never responded. He just wasn't interested anymore.

Brock: Well, it's a very collegial thing of you to do, was--

Meyer: Well, I didn't feel like I was giving away undue credit, but it was something that kind of grew out of all the ideas that he had worked out and that I had worked out. But the second paper was a technical paper that was sort of saying, "Hey, once you get the idea that all of this syntactic hierarchies you're talking about is how much time and resource things take, well, once you're talking about resources, there's lots of weird ways to put limits on how much resource, wavy functions of a kind that aren't so nicely uniform. And they'll create hierarchies, too!" So the next paper was about how you could have classes that were like this hierarchy classes, but neither one contained the other. It wasn't that one was bigger than the other. Each of them had functions the other didn't have. And they went all the way up! So that once you got the idea that, "Hey, you know, the syntax is really just talking about computational resource." Computational resource is something you can easily see lots of ways to manipulate.

Brock: Oh, I see.

Meyer: And make choices of, and then you could come back and have these so-called hierarchies that are unlike any of the other hierarchies.

Brock: Oh, that's interesting. It's a novel kind of hierarchy of functions if you did something kind of unusual with the resource and distinction. Like throttling it up and down.

Meyer: That's right. So you could get them so where they throttled up and down out of sync. So one is big when the other is little, and vice versa. That means that each one of them can do things when it's big that the other can't do.

Brock: Oh.

Meyer: But neither one does what the other one does. So that was one idea that you could have these incomparable classes and build hierarchies out of them, and all kinds of stuff like that. So that was what the second paper was about. And to the best of my knowledge, Dennis never read it and wasn't interested in it.

Brock: Huh.

Meyer: Now a reference that I was going to try to chase down on your behalf, but I will leave to you is-- when was this? At some point, when I was-- maybe when they had that celebration in honor of Dennis after he died.

Brock: Yeah, at Bell Labs.

Meyer: Yeah, of-- but I think it was even earlier than that. Well, I came upon a published interview with Dennis that was published, I don't know, some findable place-- *New York Times, Science Magazine,* something like that, in which Dennis was being interviewed by his own background, and how got into the systems stuff. And he says something like, "Well, you know, I was a graduate student in theory at Harvard, but that experience made me realize, when I finished my Ph.D., that I wasn't smart enough for theory, and so I shifted over into systems." And I like to say that my major contribution to computer science was driving Dennis out of theory! By convincing him that he wasn't as smart as me, and that, you know, he ought to go work where his talents lay--

Brock: Build something. Yeah.

Meyer: -- instead. I have no idea whether that's really true, but it's a good story. I can't-- and you'll find a quote in a passage that--

Brock: Yeah, I'll look.

Meyer: Saying that he wasn't smart enough to do theory, so he shifted over into systems.

Brock: Isn't that interesting?

Meyer: Yeah.

Brock: And do you think that's why-- I mean, there's-- in speaking with his surviving siblings, you know, who tracked down and found a copy-- or who found a copy of his thesis, his copy of his thesis, and tried to track it down with Harvard University at status and found--

Meyer: Did I tell you that story?

Brock: No.

Meyer: Or do you want that one?

Brock: I would love to hear that story.

Meyer: That's another good story. This is how true we-- well, we were just finishing up something else you were going to say. So you want to--

Brock: No, we'll get back to it. I would love to hear that now. Yeah.

Meyer: So the story as I heard from Pat Fischer, I believe, told this to me. I'm sure that it was Pat who told it to me. I've never had it confirmed elsewhere, though, but I believe it, was that it was definitely true at the time that the Harvard rules were that you needed to submit a bound copy of your thesis to the Harvard -- you needed the certificate for the library in order to get your Ph.D. And as Pat tells the story, Dennis had submitted his thesis. It had been approved by his thesis committee, he had a typed manuscript of the thesis that he was ready to submit when he

heard that the library wanted him to have it bound, and give to them. And the binding fee was something noticeable at the time. I forget whether it was, you know, $50, $250, something or other. But it was a-- not an impossible, but a non-trivial sum. And as Pat said, Dennis' attitude was, "If the Harvard Library wants a bound copy for them to keep, they should pay for the book, because I'm not going to!" And apparently, he didn't give on that. And as a result, never got a Ph.D. So he was more than everything but thesis. He was everything but bound copy.

Brock: EBT.

Meyer: But he apparently drew the line at that. It was a matter of principle, and didn't get a Ph.D.

Brock: Oh, I guess it just shows how kind of a self-contained creature.

Meyer: Or something!

Brock: Or-- yeah, and who kind of-- as a result, I don't think--

Meyer: Didn't hurt his career any.

Brock: It didn't, no, but as a result nobody really has-- I mean, I guess they've seen in your co-published paper a lot of that.

Meyer: Well, the co-published paper is not really what was in the thesis. The thesis was his version of--

Brock: Of solving that--

Meyer: -- of solving the problem. Yeah.

Brock: -- problem. Okay. Okay. Okay, wow, that-- and what--

Meyer: And my version, by the way, was in the joint paper about Loop programs, because by the time that that was done, I was already doing lots of other stuff. And I think that this was not part of my thesis.

Brock: Oh.

Meyer: His work.

Brock: Oh, right, because you-- now did you-- in your thesis did you extend the use of Loop programs, or did you use the concept of resource?

<overlapping conversation>

Meyer: No, no. As I remember, I didn't use any of the joint work.

Brock: Okay.

Meyer: My thesis was about other aspects of complexity theory.

Brock: Okay.

Meyer: And computability theory. Maybe the second paper might have been in my thesis. I should look that up, but I don't remember. I don't think so. I think that the work with Dennis was going to be his thesis; and I didn't need it because I had lots of other stuff already. But--

Brock: How did-- now it seemed to me-- well, just to finish the Dennis Ritchie thread, did you have any contact with him over the years after your--

Meyer: No, not really.

Brock: -- time together?

Meyer: I remember arranging a visiting-- a lecture invitation when I was in my first or second year after we graduated with our Ph.Ds. I was at Carnegie-- it started as Carnegie Tech, and the following year it became Carnegie Mellon University, when they acquired the Mellon Institute.

And I arranged for Dennis to be invited as a speaker, because, you know, I was impressed with him and I liked him. And I thought that-- I'm not sure exactly what I thought, maybe just because I'd like to see him, or because-- the department might be interested in making him an offer, or whatever, or maybe he was just doing interesting stuff. So he came and gave a lecture on what at that time was B, not C.

Brock: Oh, uh huh.

Meyer: And I remember-- of course, it didn't mean anything to me, because I was wary of real programming language stuff. I didn't do that. But I remember Alan Perlis, who was the department head at that time, who was maybe the most distinguished figure in programming languages -- at least American figure in the programming language business -- and he said he was very unimpressed, and he just didn't think that there were many important ideas in this B language, that it was going anywhere. So Alan didn't get that one right. Because B became C, and C became a core aspect of UNIX, and remains for decades, generations, an enormously important and influential programming language. So I had positive enough feelings and was interested enough in what Dennis was doing, felt it might be important to arrange that interview, and an invitation to come to speak at Carnegie. But then it didn't go anywhere, because Perlis wasn't interested, and I didn't really know anything about that idea in order to defend it and say Perlis was wrong. So it dropped and that was the end of it. And I don't think I had any further contact with Dennis after that.

Brock: Well, could we spend a little bit of time talking just-- I had some questions just about, you know-- not "just"-- about the arc of your career.

<Break>

Meyer: And maybe we can invent some other ways to explain what's happening. But it was a great-- again, it's an illustration of this intellectual concept which I thought, and was tremendously excited by, at the very beginning of my career, which is that computation is a universe-- a crucial, almost universal lens through which to view other problems. And that looking at things through a computational lens suddenly reveals phenomena that had been not even noticed by the practitioners in the field. Equilibrium in economics is a wonderful example.

But I remember early on one of my own results was about, in algebraic geometry, I didn't even know what algebraic geometry was exactly. But there were these various results about behaviors of polynomials, and when we looked at them as computational problems, we could prove results that the algebraic geometers had never even thought of proving! And they were very interested in when we pointed it out to them by saying, "Well, you know, let's not think about them as numerical things, and ask about the varieties and shapes geometrically, let's just think of them as, well, suppose I used a compu-- a polynomial expression to compute with, what sorts of values could I get it to compute?" And when you ask that, you suddenly got into a way to apply all the computational theory that we could bring to bear.

Brock: Yeah, it's an interesting-- I can see how it'd be an interesting new perspective to bring to bear on different areas of mathematics, but also different areas of culture that use mathematics to get at various things about the human world, the natural world.

Meyer: Mm hm.

Brock: Yeah.

Meyer: Yep.

Brock: It is that when you got into this computational complexity and computability in applied mathematics, it was like a subfield of a subfield in applied mathematics. When-- I'd be interested to hear you talk about like when and how computer science becomes kind of disciplinary-- like the disciplinary history. When does it really start to become distinct, when people say, "I'm a computer scientist," instead of saying, "I'm an applied mathematician"? And does the stuff that you were working on in terms of what we were just talking about with the loop programs and computational complexity, does that-- is that kind of more at the center of what was going on with theoretical computer science, as opposed to being just kind of a sub-section of applied math?

Meyer: Well, basically, the Loop programs are historical-- not artifact, but an historical event that are no longer relevant, nobody thinks about it.

Brock: Yeah.

Meyer: They're not important. What was important as this idea of resource bound and computation which remains a central theme in modern complexity theory and computability theory. I'm not quite sure how to answer the question except that I always felt that I was lucky to be doing what I was doing when I was doing it, because it was this brand new field where very little had been worked out. And a smart person who was not a serious scholar, who wasn't willing to spend years learning all sorts of stuff in order to be able to get to the research frontier-- and I've never done that kind of thing, and I don't have the patience for it-- but you could look at it and the odds are that very little work had been done, and you could immediately do it better. And that was exciting to me, tremendous source of satisfaction. So when I started off, there wasn't all these different some disciplines. There was theoretical computer science. And my early work with collaborators, we worked on algorithms, we worked on automata theory, we worked on algebraic automata theory, and we worked on time-bounded computation, and we worked on the syntactic issues in programming. And we did them all. You know? I have early publications on shortest path algorithms and graphs, and algebraic decomposition of automata theory. And we did some other work, highly technical work on the complexity of multiplying integers. Which remains an open problem, actually, to this day. People just don't know how hard it is to multiply two integers. There are methods known that are way more efficient than the nutty way that you learn in elementary school. But as of this moment, there are I think no non-linear lower bounds, knowing on how hard it is to multiply. Nobody can prove that multiplying is harder than adding. Although we have, but we had some technical results that put some limitations on the way you compute it, that it was a little bit harder than linear multiplying.

Brock: Hm.

Meyer: So that was an example of the stuff that we were doing. And at that time, working with my collaborator at that time, Mike Fischer, who was the younger brother of Patrick Fischer, my supervisor.

Brock: Oh!

Meyer: And Mike, you know, was a-- Mike was a graduate student a year behind me, very smart and a wonderful guy to work with. And you know, we would look at the papers that were coming out in the one or-- in the two leading semiannual symposia in computability, and we'd look at a paper that appeared, and we'd study it at the seminar, and the immediate test was, "Okay, let's see what he's got and we'll do it better!" And we could do that consistently! Because it just happened--

Brock: Yeah, it was wide open.

Meyer: But it wasn't very deep.

Brock: Yeah.

Meyer: So that was my hump, and very easy to be productive. And also very versatile covering all these different areas. By the time I got the-- what I felt was my own pinnacle results, the results that had accomplished what I always wanted to accomplish, and I, at that point, had a kind of maybe-- maybe it was a crisis of confidence that sort of, "Oh, my god! This is amazing! I've actually done what I have dreamed of doing." It was really very lucky, because all the pieces fit together. I happened to be the right person at the right time. And, "I'll never do anything this good again! So do I want to continue in this field where everything I do is going to be not quite as good as that? It's time to change fields!" And I switched from working in complexity to working in logic and semantics of programming. This was in the late '70s.

Brock: Which-- what was the result that you felt was that kind of crowning achievement in that area?

Meyer: What was it I felt was my crowning achievement? People, by that time in the early '70s, '72 -- 1972 was Steve Crook's paper defining the concept of NP Completeness. And that became a center of research. But it was also the case that we had all sorts of these abstract theorems about how more time gave you the ability to do more stuff.

Brock: Yeah.

Meyer: We had no concrete examples of real problems that were not just contrived by these diagonal constructions of something real that was hard to compute. There weren't any examples. And I found the first ones, and that was my major achievement and tremendously exciting.

Brock: How did you find them?

Meyer: It was-- it's a story of itself maybe we could go into, but it was just, I would say, a combination of luck and brilliance, of putting together a whole bunch of pieces that I happened to have learned about in my education, taught by teachers who may or may not have even seen that they were relevant, but they would fit together, but somehow or other they thought they were relevant, and they all came together, and bang, I had two results that were exactly what I had been looking for. One of them was-- or better yet, I'm not sure which one is easier to explain. There was this notation that had to do with what you could do with finite automata. It's a notation that's still used actually in programming language to describe-- it's a natural way of describing certain kinds of patterns of strings of dig-- strings of characters. It's called regular expressions. I don't know whether that's ever…

Brock: I've heard the expression, regular expressions, but--

Meyer: Well, you know, when you're trying to do like you're looking in a textbook, and you want you to search for certain kinds of patterns, you might sort of say things like, "Okay, I'm looking for a name that starts with some number of A's. I'm not quite sure how many-- at least one, but maybe two or three, and followed by an unknown amount of stuff, but it always ends I-N-G, or S-O-N, or something," right? So yeah, there's a little notation that says, Okay, you're right. “A* alphabet* I-N-G or S-O-N.” And that's a description that says where the star means any number of occurrences.

Brock: Right.

Meyer: So any-- so “(whole alphabet)*” means “anything at all in here.” A* means only A's, but zero or more A's. So see how you can write that little expression that describes that set of strings. And it's very easy to implement the scan that will look at some large text document, and find all the segments of a document that match exactly that scan, or that pattern. So it's a pattern

with description, right, and it's a very concise one, and widely implemented. Most programming language use operating system languages to support this. And the explanation behind it is one of the early beautiful results in finite automata theory, which is there's a certain set of notations that you could use involving-- taking-- if you have an expression that describes one set of patterns, and another expression that describes the second set of patterns, if you put them next to each other, the two expressions, that you interpret as a description of all the strings that have, begin with the thing that matches the first pattern, and continue with the thing that matches the second pattern. It couldn't be a more natural idea. Okay, so concatenation of expressions.

Brock: Got it.

Meyer: And concatenation of patterns. So you have concatenation, this star operation, which means any number of these can occur. And an "or" operation, which means it could be this pattern or that pattern. Okay. You could take the whole thing and put a star on it, which means, "Okay, it looks like some number of repeated occurrences of this pattern or that pattern. Maybe it's all this pattern, maybe it's all that pattern, maybe they alternate. But it's only a string that is a sequence of these patterns." Okay. So the beautiful theorem was that what's describable by this particular set of regular expressions that just involve union concatenation and star, are exactly the languages that can be recognized by finite automata. Beautiful example of a syntactic characterization of something that was otherwise characterized by a computational mechanism. And an early result, Rabin and Scott got a Turing Award for working this out in a very beautiful and general way, back in the '70s, probably. Okay, so it turns out that-- actually one of my results, now that I think about it, but it was sort of a minor throwaway in the context of the later results was that determining whether or not two of these regular expressions describe the same set of patterns was actually a harder problem than the NP problems. Sounds easy, but it's actually harder. As a matter of fact, just telling whether one of these expressions describes all possible strings is harder than one of these NP problems. Or at least it's a sort of doable in space that's-- and actually technically, it's linear space. Okay, maybe not deterministic linear space. I forget. But it doesn't matter which space, okay. But then what I realized, and it came out of some work with a student, an absolute genius student, but that if you add a simple abbreviation to this notation, namely, instead of writing a pattern and putting it next to itself and saying, "I'm looking

for strings that consist of two occurrences of the pattern, one followed by the other, I just write "squared.""

Brock: Okay.

Meyer: So I write pattern.

Brock: Squared.

Meyer: Abbreviation squared, it means two occurrences, of the pattern. That's the only abbreviation in the whole notation. All I'm adding to the notation of the operations of union, concatenation, and star is squaring. I can prove that that problem is exponential. Okay. So it was-- in a certain sense, it was the first example of a real uncontrived, not described by some process of diagonalization, but a very simply, easily described problem you could point to and say, "Here it is. It's easy to understand, it's easy to see how to compute it. You just build a finite automata that does the job. But hey, look! Every time, because you could have missed its squarings, this expression that looked like two to the two to the two to the two is describing a finite automaton that takes an exponential number of states, and by the way, you can't beat that!" That was the insight. You can't beat that. Okay? That it absolutely requires exponential effort to determine whether or not one of these expressions with squaring describes all possible strings.

Brock: Hm.

Meyer: And that was the mind-blowing result that I wanted. You know, it solved a real problem that we could finally say, "Here's an example, our methods allow us to prove that this obviously computable, routinely computable problem is one that's-- that pragmatically it's hopeless. That is exponential." And then related to that, once we knew how to do that kind of thing, there were a whole bunch of other problems that we could knock off similarly.

Brock: That were-- had a family resemblance to this.

Meyer: Well, the simplest one would be to take this one, forget squaring, just add complement. So it's well known that the languages that can be recognized by a finite automaton, if you can

recognize it by saying Yes or No to it, you can obviously recognize it by saying No or Yes to it. Therefore, if you can recognize something with a finite automaton, you can recognize and know strings just as well as you can recognize-- the definition of recognizing says that you say Yes to the Yes strings, and No to the No strings. Well, that means that if I just switch the roles of Yes and No, I can say Yes to the No things, and No to the Yes things, which means I am, in effect, saying, "Yes, to the things that aren't in the pattern."

Brock: Mm hm!

Meyer: So it's describing the complement.

Brock: I get it. I get it. I get it.

Meyer: So you just add that operation of complement, and again, it's trivially computable in the same way as before, because you're just building a finite automaton that does the job that the pattern recognizes or specifies. But now you discover that the “Nots,” in order to convert that, an expression that works, an automaton that works on one expression, when you have to apply it to the Not, the automaton, the Not automaton blows up exponentially. The reason was is it's a little bit harder than just saying Yes or No. If the original automaton, the usual way that these automata work is they're so-called non-deterministic automata, which is that they really have lots of different ways that they can act on a string, and the string is said to be accepted if there is a way to say Yes.

Brock: I see.

Meyer: Okay? And that sounds like one of these NP complete things, because they don't know how to say No.

Brock: Yeah.

Meyer: But it turns out that there's a standard construction, one of the basic facts in the elementary theory of what finite state machines is. You can take one of these non-deterministic automata and convert it into a deterministic one that plays in exactly one way and definitely says

Yes or No. But the conversion is an exponential blowup. To take a nondeterministic automaton and convert it into a deterministic one involve-- requires an exponential blowup in states in certain cases. That was a well-known result. But that was just sort of, you know, one finite automaton description in terms of another finite automaton description. And finite automata have already limited kind of descriptive mechanism and computation mechanism.

Brock: Yeah.

Meyer: But so the naïve way of working with one of these pattern matching things and trying to figure out how it behaved when you were allowed to talk about complements, you had a non-deterministic one, you complemented it, it blew up exponentially. Now you did some more stuff, became deterministic, but you did some more stuff and became non-deterministic again. And then you complement it again, it'd blow up again. And the result was that when you take these regular expressions extended with complement, now you can prove that this naïve approach can't be beat. That basically the complexity of telling whether or not a-- one of these non-determinant-- one of these regular expressions with complement describes the set of all patterns. Grows not exponential, not doubly exponentially, but like a growing stack of exponentials.

Brock: Oh, yeah.

Meyer: It grows like two to the two to two to the two to … height N … to the Nth power.

Brock: Uck! Yeah.

Meyer: I mean, numbers that vary-- for very small N. Actually, we have some example in a joint paper with student, Larry Stockmayer, where we did a similar example in saying, "Okay, suppose you're going to build a circuit that correctly answered whether or not a formula's in a certain logical language of length, 2000 of length, that could be expressed in 2000 bits, if you coded them into bits."

Brock: I think I was reading this paper.

Meyer: And the circuit would, if the transistors were built out of protons, and you ignore the size of the wires, the circuit would fill the known universe. Okay. So just building a hard-wired circuit to solve this finite problem of whether or not these 2000-bit formulas were true or false, was again, an example of an uncontrived problem, logical language, it so happened, a known decision procedures to tell whether or not a formula in this language was true or not. But computationally, it's infeasible, you can't do it. Okay. So those were the results that I got in 1975 working with this student, Larry Stockmayer, and then we brought in a bunch of other students. And those were the pinnacle results that I wanted. And I thought that the purpose of sort of complexity theory is to have some real problems that people know how to do and want to do, and you can point to and say, it really is hard. Our methods enable us to prove that there is no way to do this efficiently. That it's exponentially-- or worse. And once I had that, it was like, okay, well, what's next? Well, the important thing, and what people still care about, is the NP-complete problems, which remain open. But my feeling was, you know, I did what I wanted to do. I mean, yes, it would be wonderful to understand the NP problems, it's maybe more important, but I wanted to show that all this theorizing about abstract complexity theory was real, that it would get at some problems that were not contrived and made up to be hard, but preexisting. And we had a rich set of those. Lots of different logical theories that were-- systems of logic that were known to be decidable. Lots of problems related to automata, and then the stuff I was telling you about with polynomials, and their complexity, and the degrees of polynomials that define certain kinds of ideals in algebraic geometry, and stuff-- oh, and then the membership problem for communitive semigroups, another algebra problem. We knew how to get at these problems, and it was a huge number of them, where we could suddenly say to the guys doing communitive algebra, "Hey, you never thought about this. You thought you were finished with the problem when you showed it was decidable, but did you know we can prove that you can't decide it efficiently?" Yeah. So that was what-- those were the results that, by-- in '75, with a bunch of students, and we kept working on that for a couple more years that I had. And by '79, my feeling was, okay, well, I could keep working on NP-completeness and those problems, and probably continue to be a useful contributor. But, you know, the excitement, the magic, wouldn't be there. I don't think-- I can't think of anything I could do that would match this, except maybe solve the NP problem. But I don't think anybody-- nobody has laid a hand on that yet. So I changed fields.

Brock: May I ask you two quick questions about that? One is, how-- you've mentioned diagonalization several times. And I'll confess, I'm ignorant about what that is and how that works. But could you maybe, just for me and others, contrast the kind of examples or problems generated using that method, versus the kind of real-world example that you-- examples that you came up with in the mid-70s?

Meyer: Okay. So let me see whether I can pull this off.

Brock: I'm sorry if that's unfair-

Meyer: No, that's fine, no, it's a good expository challenge. So, the idea of diagonalization comes from an argument, due to a mathematician named Cantor in the late-19th century. Late-1900s-

Brock: No, late-19th century.

Meyer: Late-19th century, 1890s. And what he was arguing was, suppose you're looking at functions, again, say for definiteness-- from the non-negative integers to the non-negative integers. And you want to know, is it possible to write down a list of them all? Could you write them down so there was a function, call it number zero, and another function, number one, and another function, number two, and in such a way that the only things in your list were these functions from non-negative integers to non-negative integers. You could make it even simpler, just the ones that go from non-negative integers to zero or one. So these would be kind of the yes/no questions, okay? And such that every such function appeared in your list, yeah? Eventually, somewhere. And only those appeared. Is that possible? And Cantor's answer was no, no, very simple argument, diagonal argument, showed that you can't do that. So this was saying-- the first result that says there are-- it's not countable-- there are not a countable number of functions on the non-negative integers, even into zero, one, because the definition of countable was you can write them in this list.

Brock: Got it.

Meyer: You can count them. This is zero, the first one, second one. You can't do it. Okay? And-

Brock: Too many.

Meyer: There's too many, yeah. And there's a very simple diagonal argument that proves it, and the argument is-- if you can visualize this, I'd do it on the board for you, but probably you can visualize it. Think of one of these functions. Remember it's going to-- it gives the value of zero or one for every number. So it's either zero or one at zero, and it's zero or one at one, and it's zero or one at two, and it's zero or one at three. So you can represent such a function by an infinite sequence of zeros and ones. So the *n*th element, the *n*th bit in the list is one if the function's one at *n* and zero if the function's zero at *n*. It's just the value of the function at *n.* So every one of these possible zero or one functions is simply an infinite sequence of bits. A row, yeah? So, now, what Cantor is asking is, okay, can you have a rectangular matrix that's infinite to the right and infinite down, where the first row is an infinite sequence of zeros and ones, the second row is an infinite sequence of zeros and ones, the third row is an infinite sequence of zeros and ones. So it's just this two-dimensional matrix of zeros and ones, but it's infinite down, it's infinite to the right. Can you have one of those where every infinite sequence of zeros and ones appears as a row? That's the question, yeah? Reasonable question. I mean, I can write down an infinite number of these rows. Is it possible that every-- could I possibly have every conceivable row in my particular list of rows that are numbered zero, one, two, and so on? And the answer is no. And the proof is a trivial example of one-- you give me your list of rows, and I'll show you a row that you don't have. And the row that you don't have is defined as follows-- go down the diagonal and flip the bits. Take the infinite sequence of zeros and ones who's *n*th bit is the opposite of your *n*th row at position *n*. It's just different. So, by definition, your diagonal sequence of zeros and ones is not a row, because pick a row, the 55^{th} row, it differs from in position 55. It's not 107, because that's different from your 107^{th} row at position 107. End of story. I've just described to you an infinite sequence of zeros and ones that's not a row of your list, okay? That's a diagonal argument, because you're going down a diagonal of this matrix, flipping the bits from zeros to ones.

Brock: And this became like a paradigmatic-

Meyer: This is a paradigmatic argument for how do you, when you have a countable list of things, numbered zero, one, and so on-- how do you find something that's not there? It's easy to

do. You just go down the diagonal. And of course, you don't have to go down a diagonal, you could go down a 2/3 diagonal. You get another one. You could even have ones where you differ in a complicated pattern, and you not only differ from everything in every row, but you differ at infinitely many places from every row. And so on. It's just-- you can play all sorts of stuff with it. That was diagonal arguments.

Brock: Okay. And this was an example of a very-- so how does that become-- the diagonal arguments become a model example of something that's hard to compute?

Meyer: Well, they're the basic idea for how do you find something that's not there. When you give me a countable list of functions, how do I find one that's not there, okay? So, now, there's another-- now, this is the idea of how it applies to complexity theory. Okay. It's very easy to make a list of all the computable functions, because a computable function is defined by a little object, like a Turing machine. A Turning machine, I can always code it to some set of bits. A set of bits is a number, okay? So I can, in a sense, number all the Turing machines. This is the zero Turing machine, the first Turing machine, and so on. And whatever the function is that's computed by the *n*th Turing machine, I'll make that row *n.* Assume that we translate-- whatever the output of the Turing machine is, we translate it into zeros and ones. Or, the thing about Turing machines that makes it slightly complicated is that it might not give an answer at all. So I need to deal with that. And let's say-- I don't care what you do. Change all the place where it doesn't give an answer, make those zeros. Okay. So, now, given a Turing machine that's going to be applied to inputs that are numbered zero, one and so on, with every Turing machine, I can associate this computable function of zeros and ones that it produces, and I definitely can list all of those, because the Turing machines are little, finite bit strings, and I can list them one after another by length of bit strings in alphabetical, and so on. So, in short, that's a hand-waving argument that says, the computable functions can be listed, by listing the Turing machines. Because every one of them is computable by a Turing machine. You can make a list of all possible Turing machines. Okay. That immediately gives you a way to get functions that are not computable, because the general diagonal argument says, give me a list of functions, here's how to make a new one. The computable ones are list-able, so there's easy to find non-computable functions. Okay? Easy. Same thing applies, if you talk about the functions that are computable within a given timebound. Suppose I'm interested in the functions that are computable in time n-

squared, okay? How do I do that? Well, if it's computable in time *n*-squared, it's computable by a Turing machine that always stops after *n*-squared steps. How do I make a list of those? Well, it's easy. Take all the Turing machines, and put, on each Turing machine, a monitor that shuts it off after *n*-squared steps. So add to the program, a thing that counts what the program is doing, and stops after *n*-squared steps. It computes *n*-squared and only lets the other one for that many steps. So I can take every Turing machine and put a little monitor on it, and the monitor won't have any effect if the thing already stops at time *n*-squared, and if it doesn't, it forces it to. The result is, I've just proved to you that I can list all the programs that run in *n*-squared steps. And that means I can, by diagonal arguments, find one that's not doable in *n*-squared steps. But, the key thing is that in fact, when I'm trying to figure out what's on the *n*th diagonal, well, I have to figure out, what's the *n*th machine in my list, and I have to run it for *n*-squared steps. Well, simulating it may cause a little overhead that it takes a constant times *n*-squared to simulate it. But with a little bit more than *n*-squared steps, I can simulate it, which means that the function, the diagonal function, can be computed in just a little bit more than *n*-squared steps. So, suddenly, by a diagonal argument, I've shown, ah, give me a little bit more than *n*-squared time, I can find something that can't be done in *n*-squared time. A little bit more can actually be quite-- little more like log of *n*. So there's something that's computable in *n*-squared log *n* steps that are not computable in *n*-squared steps. Okay? So routine extension of those 1930 ideas in diagonal arguments, okay?

Brock: Okay, and so these were the kinds of model problems, examples, used in the-

Meyer: Well, these were paradigmatic solutions.

Brock: Solutions to the questions.

Meyer: These were said that-- these indicated, how do you carry over the 1930s proof methods and concepts of yes or no-- can be, or can't be, computed. We can routinely carry those over into: Can it be computed fast? Can it be computed in *n*-squared or *n*-cubed? And we know more time gives you more, by a diagonal argument, which I just sketched for you. So that was all well, and then we did all kinds of cool stuff with sophisticated diagonal arguments, and proved all sorts of things, but it was all this-- what they accused at the time of being "Cantor's paradise,"

the mathematicians thought that he was, you know, he's theorizing about a world that nobody cares about, it's all abstract and there's nothing real about it. The same issue was what plagued me and other people, which is, we have this wonderful theory of you can do more with more time, but what? What was it that you can do more? Don't tell me it's this diagonal function. I don't care about that. I mean, yeah, you show me this diagonal function that I can simulate all the slow ones with a slightly slower one, and therefore I can do more, well, big deal. It's not interesting, okay? It's a theoretical result that doesn't give me anything concrete. And the connection was that when we learned how to recognize, in various real problems, their ability to describe computations that took a limited amount of time, that meant that those problems required that much time to solve. Okay? That was the connection that was made. So that was the thing that, you know, I remember being criticized by giving one of these abstract complexity theory talks, and a very famous logician, Hilary Putnam, was in the audience. And he said, "What does this have to do with checking satisfiability?" The NP belief. And I said, "Well, it doesn't really have anything to do with that that we can see. These are abstract results." They shrugged, and politely walked away. So could we connect up all these abstract ideas with some real problems? And that was what I did. And it was the kind of pinnacle result of my career that I felt, Jesus, I'll never do one like this again.

<Break>

Meyer: You were asking about how the field had grown and changed into things--

Brock: Yeah, become computer science.

Meyer: Yeah. So, what happened was, and it was just starting to happen at the time that I decided to shift fields altogether, was that by the late-70s, this tiny subject of theoretical computer science and theory of computation, which involved a few researchers who did everything, suddenly grew into a significantly larger community of researchers, five, ten times as large, and began breaking up into disciplines. So they came to be algorithms and semantics and complexity theory and logic. And each of those got to be specialized enough that they started having their own symposia, and there are now, I don't know, probably a dozen or more different annual technical symposia that represent papers in these sub areas. And all of the sub areas, now,

are many years deep. So that to get to the point where you're doing forefront research and solving open problems, it takes a couple of years of reading and study to catch up. And it is no longer the style that I enjoyed and I was good at. So I switched from this sort of general area of complexity over into doing logic and semantics. And I think I-- it's fair to say that I got to be a visible figure, and made some modest, basically expository contributions, but never was the kind of intellectual force that I had been earlier. But that's sort of the way it went. And I continued to have a productive and enjoyable research career for about 25 years, I guess, until I got to be 50 years old. And at that point, I stopped doing research. I decided I just couldn't do it anymore. I felt like I wasn't-- something was no longer working for me. I would still seem like a smart, quick guy, but I knew that it wasn't the same kind of thing. And that I wouldn't be fair to work with students anymore, because I couldn't give them the same kind of commanding vision and insight that I had previously. And I think that that's right. I mean, you never can be sure about a judgment like this, but I think it was sound.

Brock: You stopped taking on graduate students?

Meyer: I stopped-- at age 50, after a very productive research career, in which I was a, you know, world-famous and influential figure, certainly in complexity theory, and to a lesser degree, but still noticeable, in semantics and logic, I decided that I just couldn't do this anymore. Maybe I couldn't or I wouldn't. That I was tired of living a life in which I basically wasn't fully there, most of the time. Because I could be standing on line in the supermarket, talking to people, and I would really be proofing theorems. Okay, I mean, that's what I was doing, and that's what I was obsessed with, and thinking about, all the damn time. And I got to the point where I just couldn't do it anymore. Burned out. And so, you know, so I spent another 25 years as a professor. The nice thing about being a professor is you're a professor forever, they can't fire you. So just out of trying to earn my keep, I spent a lot of energy on teaching, and I took on a whole bunch of academic administrative responsibilities of, you know, being the head of the graduate program, head of the undergraduate program, secretary of faculty, all kinds of middle-level management things, academic management. Plus working a lot on teaching an introductory course in discrete mathematics, and we've written a 1000-page textbook that we have yet to publish, but it's been widely adopted and used, because it's available for free online. So that's what I have done kind of professionally. But I haven't done research since I was about 50 years old.

Brock: And while you were taking on graduate students, about how many students-

Meyer: 28.

Brock: 28?

Meyer: Yeah. Which I didn't realize at the time, but it was a very unusually large number. I always felt that it was kind of a slowing down, inadequate, that I was only getting about one Ph.D. a year. Because there were some years-- there was one year where I think I had six PhD's finish, that year when we got-- when we learned how to proof all these problems that were hard, suddenly there was just an open-ended amount of effort from lots of people to develop all of this. So it was very easy to find thesis topics for all these things. But-- so that was an accomplishment, and it was very nice, when they had a retirement party for me a couple years ago, that many of them showed up and talked about what a wonderful supervisor I was. And I guess I was. But it was just that period of my career. And then I learned another thing, which is, I thought that I had always loved-- I did love what I was doing, and was tremendously enthusiastic about it, excited about it, and I communicated that excitement and enthusiasm to my students and others. But, again, it's a poignant realization, it wasn't that I liked the subject. I liked being good at it. And once I felt that I wasn't as good at it as I wanted to be, I didn't want to do it anymore. So it wasn't a matter of, you know, I undoubtedly could have continued to be a competent, modest contributor. But not in the way where I was producing results that people would say, "Oh my god, that's amazing, astonishing, beautiful," and so on, which is what I had done, repeatedly, earlier in my career. And that was what I got satisfaction out of. And once that was no longer a realistic aspiration, I, in fact, lost interest. So it's interesting now that in my retirement I'm just going back and looking, again, at some of the amazing work in complexity theory that has gone on since I stopped paying attention. And I think it's fair to say that some of the stuff, the early stuff that I contributed, continues to be very influential. But what the field has become is really quite amazing and impressive, and I'm starting to learn it again, just out of curiosity. I don't expect to be really working on it, aiming to make contributions, but I just read an absolutely masterful, extensive survey of the field by a junior colleague named Scott Aaronson, who wrote the most extraordinary comprehensive monograph on the state of research on P=NP two years ago. And in fact, I read it and was tremendously impressed with it, but there are a couple of

things in there that I think aren't quite right. And which I am-- I need to work them out a little bit more, but I plan to write to Scott about it. Well, one is a misattribution of a theorem that's actually due to me that he credits to Steve Cook. But the other is a result that's widely cited, about optimal problems for NP-complete problems, that I think is misleadingly stated and actually wrong, in the sense that there isn't an optimal program. Although it's widely said that there is. There's an optimal recognizer, but not acceptor, which is a technical distinction. But it turns out-- it gets the yes answers in about the best possible time, but any given program will do badly on infinitely many no answers. That's unnecessarily large amounts of time on some no answers. You can look at it and figure out how to make it better on and infinite number of no answers, which is very cool. And an example of a so-called speedup phenomenon, which was another thing that we were very concerned with showing was real. There were these-- out of diagonal arguments came this weird phenomenon that there were certain programs that were computable. But it was no best way to compute them. Given any program that computed them, there was another program that did much better on all but finitely many inputs.

Brock: And I think I saw some result that you published with Stockmeyer? That there is-

Meyer: Yes, well, not with Stockmeyer, although there was one related with Stockmeyer. There's stuff published with Robbie Moll, and on effective operator speedup, and I'm not sure what other speedup papers I may have written. Did some related stuff with Mike Paterson, but this was-- the basic result about speedup was due to Manuel Blum, who proved this really remarkable speedup theorem. He showed, for example, pick your favorite rapidly growing function, let's say two to the *x*, okay? And we'll say that one function is much slower than another function. If it's-- it's at least as big as two to the first function. So two to the f is much bigger than f, okay? Okay. So there are certain kinds of computable functions, where any program that computes them, whatever amount of time that program takes, there's another program that also computes them, and the one you had is much bigger than the one that I have. So my program, in effect, takes the logarithm of the amount of time of your program. But it keeps going, because now that I got one that's log faster, I can beat that by a log. And you can beat that one, I can beat that one by a log. So any finite number of logs, I can keep making them better and better and better. There simply is no best program. For any program, there's a much better one.

Brock: That was the result that I saw, just that way that you put it. That really-

Meyer: It's a mindblower, okay? So the puzzle there is, there has yet to be, and may never be, a realistic example of a problem where that happens. It's one of these things where you can make it happen by a diagonal argument, but all you have is, okay, this artificial example that I can build by a diagonal argument has this property. Does anything real have the property? And no one has ever figured out how to make that happen with a real example. But there's a weaker kind of result, but it's got a similar flavor, which is, Blum's speedup is amazing, because the speedup, the new program, is much faster than the old program, at all but a finite number of places. It really is exponentially faster, except maybe at the first million inputs. For the first million inputs, you can't improve it, because the first program does those inputs instantly, the second program can't do better than instantly. But once the first one gets going and starts to take a very large amount of time, I can beat that. But a weaker kind of speedup but still interesting is something where it doesn't happen at all possible large inputs, but just it happens infinitely often. It still means whatever program you have, I can find another program that, on infinitely many different inputs, I'm much faster than you are. So-- and of course, that one, it's another program. So I can beat it in infinitely many places. And so on, all the way down, and it still means there's no best one. But they're not best, just getting better everywhere, they're just getting better in an infinite number of places. It still means there's no-- doesn't make sense to talk about a best one. Okay? That's a real phenomenon. And that was what was proved by jointly with my student Larry Stockmeyer, and it appears in his thesis. First of all, the existence of problems with this kind of infinitely-often speedup-- of course, that's immediate, because it's a weaker result than the everywhere speedup that Manuel Blum had. But the insight was that this infinitely often speedup property was one that could get communicated to real problems. And you could take the abstract problem and show how a real problem, like decision procedure in logic, or some problem about regular expressions, could simulate the diagonal construction, which meant it inherited the property. So the result is that, essentially, all of the NP-complete problems have this property that we don't know how well you can do, but given any program that did them, I can get a-- I can look at the program that does them fast, and I'll find you another program that beats them in infinitely many places, okay? Because they have to be doing stupidly on an infinite number of inputs, and I can find those and suddenly find a program that's smarter than they are at these

infinitely set of inputs, yeah? So that was, again, a connection with trying to take the diagonal abstract problems and make them real.

Brock: Right, right.

<Break>

Brock: Yeah. Well, maybe I'll just ask you to-- I mean, reflect on, you know, you have been involved with what has become computer science, from its earliest incarnations.

Meyer: Well, theoretical computer science, yeah.

Brock: Theoretical computer science from its earliest incarnation. I would just be fascinated to hear you talk about how the community of people engaged with theoretical computer science has changed, stayed the same, just in terms of where it's happening, who's doing it, just what the picture of that community looks like.

Meyer: Well, I haven't-- I haven't been actively involved in that community in 25 years, so I hesitate to give an authoritative answer, or even one that I fully believe in. But what strikes me, when I go back and start to-- as I said, I was reviewing this subject that I left 25-- more, 35 years ago, or so, the computational complexity subject that I let go of probably in the late-70s. And what's happened since is really awesome and impressive. And I'm struck, simply, that there's what seems like an army of very smart people who have done deep, extensive work, building on each other's efforts. And all of the areas, as I said, that looked to us in those early years, like one subject, theory of computation, and what is the significance of computation, and its role in other subjects, now has turned into at least a dozen different sub-disciplines. And I haven't taken a count of how many people are involved, but it's probably at least ten times the number that was involved at the time that I was an active researcher. And there are very few people, anymore, who even try to keep up with more than a couple of those sub-areas, because they're just too rich, and there's too much going on, and too much to know about. There are a few, very few, old-timers that have continued to practice. Dick Karp is maybe a notable example, just absolutely extraordinary guy, who absolutely did it all and continues to do it all. I think he's worked on everything from abstract complexity to algorithms to probabilistic algorithms, to approximate

algorithms, and so on, and continues to be a productive figure. He must be in his early-80s now. He just retired as the director of the MSRI, Mathematical System Science Research Institute at Berkeley. So he is an extraordinary figure. A couple of others-- I guess Silvio Micali would count, I think, as one of these people who's working in multiple areas. Silvio is, of course, famous because of his work in crypto, and the notion of zero knowledge, and is a seminal figure in modern cryptology and security, code-- working jointly with Shafi Goldwasser. They got the Turing Award jointly about five years ago, six years ago. And Shafi is now the new director of MSRI. Silvio, I think, would count as one of these people who worked definitively in at least the areas of algorithms. He's got the, I think, the best perfect matching-- or optimal matching algorithm for general graphs, is due to him and Vijay Vazirani. And it's extraordinary technical contribution in that area. Has nothing, whatsoever, to do with crypto or complexity theory. And also, a seminal figure in crypto and zero knowledge. Not many other people, that I can think of, that cross areas like that. Maybe [Michael] Rabin, who's, you know, a towering genius, but-- and worked in logic and automata theory, and moved over into crypto 20 years ago, but I don't think his approach to crypto has, in fact, been a significantly influential one. So I'm not aware, now, and I may-- probably I'm overlooking some eminences, but I'm not aware of other people that would fit that pattern of world-class contributors in multiple areas. I think, even Shafi Goldwasser, who was a co-author with Silvio, I'm not aware that she's made those kind of contributions in other areas. She's made extraordinary contributions in complexity theory and cryptology theory, but I'm not aware that she's worked in algorithms or other kinds of sub-fields like that. Dick Lipton, I guess, is another one who's done everything, and is an amazing guy, and runs a wonderful blog, and has run a fascinating blog for 20 years or more. I don't know whether you've heard of him.

Brock: I'll confess I haven't, I'm sorry to say.

Meyer: He's just an amazing figure, who, again, he's been in this business for as long as I have, just about. I think he was a student of mine, like I'm five years older than he is. And I mean, literally, not a PhD student of mine, but he was literally a student. He took some courses from me when I was a young instructor and he was a graduate student at Carnegie. And he's worked on everything, and made significant contributions on everything. Just an amazing guy. Not many figures like that anymore. Maybe-- now that I think about it, maybe Avi Wigderson, at the

Institute for Advanced Study, would also count as a-- such a universal contributor. And now that I think about it, Les Valiant, who I think is still active, and is a Turing Award winner, what else did he win? He won some other major award recently. I don't think it was the Japan Prize, but something very-

Brock: I'll look it up.

Meyer: And he's another kind of guy who's worked across the board. Early work on context-free language parsing and matrix multiplication, and connecting it up to computational complexity concerns, which he got the Nevanlinna Prize thirty years ago, and a later Turing Award prize, and continues to be enormously influential. He invented the paradigm of probably approximately correct, which turns out to be a wonderful, rich, important theoretical idea. And he was the one who saw that you could take this naively sloppy idea and turn it into something crisp and beautiful and important. So he's worked on algorithms. He worked on complexity, he's worked on language theory, and been a visionary, even, in working on things like theory of evolution, theory of mind, and how these ideas might be applicable to the way memory works. So there are a few people like that, who are these innovative field builders, but not-- very few. I mean, people like that. So that's really about all I can tell you. When I look back at-- when I look more recently at this literature in complexity theory, I am impressed by how many absolutely first-rate people, by what were the standards when I was active in the field, these are countless first-rate people, and there's dozens of them now. It's amazing how many very smart people have gotten into this area and made really worthwhile and genius contributions. Those are my impressions.

Brock: That's a very posi-- well, maybe that's a very positive note to end on.

Meyer: Yeah, that's a good note to end on, if you like. Yeah.

END OF THE INTERVIEW

