An interview with
Gaston Gonnet

Conducted by Thomas Haigh
On
16-18 March, 2005
Zurich, Switzerland

Interview conducted by the Society for Industrial and Applied Mathematics, as part of grant # DE-FG02-01ER25547 awarded by the US Department of Energy.

Transcript and original tapes donated to the Computer History Museum by the Society for Industrial and Applied Mathematics

ABSTRACT

Born in Uruguay, Gonnet was first exposed to computers while working for IBM in Montevideo as a young man. This led him to a position at the university computer center, and in turn to an undergraduate degree in computer science in 1973. In 1974, following a military coup, he left for graduate studies in computer science at the University of Waterloo. Gonnet earned an M.Sc. and a Ph.D. in just two and a half years, writing a thesis on the analysis of search algorithms under the supervision of Alan George. After one year teaching in Rio de Janeiro he returned to Waterloo, as a faculty member.

In 1980, Gonnet began work with a group including Morven Gentleman and Keith Geddes to produce an efficient interactive computer algebra system able to work well on smaller computers: Maple. Gonnet discusses in great detail the goals and organization of the Maple project, its technical characteristics, the Maple language and kernel, the Maple library, sources of funding, the contributions of the various team members, and the evolution of the system over time. He compares the resulting system to MACSYMA, Mathematica, Reduce, Scratchpad and other systems. Gonnet also examines the licensing and distribution of Maple and the project's relations to its users. Maple was initially used for teaching purposes within the university, but soon found users in other institutions. From 1984, distribution was handled by Watcom, a company associated with the university, and 1988, Gonnet and Geddes created a new company, Waterloo Maple Software, Inc. to further commercialize Maple, which established itself as the leading commercial computer algebra system. However, during the mid-1990s the company ran into trouble and disagreements with his colleagues caused Gonnet to withdraw from managerial involvement. Since then, he feels that Maple has lost its battle with Mathematica. Gonnet also discusses Maple's relation to Matlab and its creator, Cleve Moler.

From 1984 onward with Frank Tompa, Tim Bray, and other Waterloo colleagues, Gonnet worked on the production of computer software to support the creation of the second edition of the Oxford English Dictionary. This led to the creation of another startup company, Open Text, producing software for the searching and indexing of textual information within large corporations. Gonnet explains his role in the firm, including his departure and his feeling that it made a strategic blunder by not exploiting its early lead in Internet search.

Gonnet continued to work in a number of areas of computer science, including analysis of algorithms. In 1990, Gonnet moved from Waterloo to ETH in Switzerland. Among his projects since then have been Darwin, a bioinformatics system for the manipulation of genetic data, and leadership of the OpenMath project to produce a standard representation for mathematical objects. He has been involved in several further startup companies, including Aruna, a relational database company focused on business intelligence applications.

HAIGH: Thank you very much for agreeing to take part in the interview.

GONNET: You are very welcome.

HAIGH: I wonder if you can begin by talking a little bit about your early life upbringing and family background?

GONNET: Well I was born in Uruguay from my mother who was Austrian and my father who was Uruguayan from French parents. I went to school and university in Uruguay and in the early 1970s decided to do graduate studies abroad.

The picture of the world, from where we were, was a little bit distorted, and it was actually very difficult to find, from South America, which were the good universities and so on. So you ended up making decisions on casual information, on hearsay. At the time there were two things that were important for anybody that was going abroad. One was the ability to fund yourself while you go abroad, and the other one was doing what you wanted to do. I wanted to do computer science; that was pretty clear. I had already an early start with computers, mostly linked to commercial applications of the IBM computers of the time. IBM was big on the University of Waterloo for teaching at that time, because the University of Waterloo had developed a system called WATFOR, which was a Fortran compiler that had very quick turn around. They were sending people from all over the place to Waterloo to show them how efficient they were. One of my professors had gone on one of those trips and he came back very impressed with the University of Waterloo. So that was one sample point of information that somehow became very influential.

The other choice I had was to go to the University of Essex in England. I had been given a grant to do a master's there by the British Council. But the British, being very cautious, were saying, "Oh, you should not bring your wife. This is very tough. We are going to give you just enough money to survive. We are going to guarantee only one year—whether you finish or not finish you are on your own," and so on and so forth. It was so intimidating that I ended up deciding to go to Canada, even though Canada was not offering a grant; the only thing they were offering me was a teaching assistantship to fund myself. But I have to say that the English scared me at the time. Also I had a misinterpretation of what is a professor and what is a lecturer and so on. As I was reading information about the University of Essex, I heard that they were all proud that they have a new professorship. So a place that is so proud that it has one more professor sort of raised eyebrows. Well, it was just lack of understanding of the system. But the bottom line is that I ended up going to the University of Waterloo in Canada.

HAIGH: Let me pull you back a little bit from that to talk about the period before you left for Canada. So as you were growing up and in school, had you always been particularly interested in science and mathematics?

GONNET: Yes, I have to say that I had always been interested in mathematics. Mathematics came easily to me. It's funny that my mother used to say, "Oh, he is good at math. This is really the dumb subject because he doesn't really need to study much to be good at math." So that was her excuse for, "You don't work hard enough. You are good at the topics that you have to make no effort for," instead of being a good student because you study a lot. I have to say that I was raised with my mother; my parents divorced when I was four and a half, so I had basically no contact with my father. So I was raised by my mother and a half brother of mine that was really my brother, who was 12 years older than I was.

It was a little bit of a hard life for me at the time, and I had to start working before going to university just to support myself and to support my family. I found a job at IBM working at first with standard equipment and then programming computers, which came relatively naturally to me, or at least was easy to do. I'm talking about the late 1960s, when computers were still very rare, even in business. At the time the computer was a 1401 with 8K characters of memory, actually 8,000 characters of memory because the 1401 was partly decimal. The Universidad de la Republica in Montevideo, was renting time from IBM at the time. I was working for IBM, but I was helping the people at the university to make good use of their computing time. I became very good friends with the people at the University who were doing computing and I was hired as an assistant. When the university decided to start a branch of engineering that would do computer science, or what was perceived to be computer science, I was there. I was sort of employee number one for computer science.

HAIGH: So when you say you were working for IBM, were you working on their behalf at the university or with a computer that was located at the IBM office?

GONNET: Actually both are true. At first I was just an employee of IBM. As a matter of fact I was called a spy in the local jargon, because when people would come and rent time from IBM, IBM wanted to make sure that they were using the equipment properly at that they would sign the time sheets and that they will not create havoc, and also to help the users a little bit to get good use of their time. That was my role when the university people were coming, to help and to monitor them and to make sure they were using the machine properly. But later as the relation with the university people developed, I eventually became an employee of the university.

HAIGH: So would you say that your primary and secondary education had given you a good grounding in science and mathematics before you went to the university?

GONNET: Yes. As a matter of fact, yes. My whole inclination for computer science has always been relatively mathematically oriented. Computer science in Uruguay at the time was developed in an environment that was very mathematical. There was "Instituto de Matemáticas y Estadística", which fostered all the computing activities. It was the best place in Uruguay to do mathematics. All the good mathematicians were there. So yes, I was a very young student, but I was surrounded by very good mathematicians that were treating me as a colleague. So that, from the beginning, taught me serious mathematics as opposed to what a normal student would do in computer science nowadays. Computer science students will have some mathematics requirements, but will be really secondary to their main education. In my case, mathematics was main line for me, so much that for quite some time I was always thinking myself as a mathematics person, not as a computer scientist.

HAIGH: Was the educational system that you would apply to university to study in a particular program, rather than choosing a major later?

GONNET: Yes, as a matter of fact my idea was to do engineering. I had several ideas of course, but my mother had this notion that I should be an engineer. She had pictured that I probably was good at engineering. Funny thing, when I went back with my Ph.D. in mathematics (it was in computer science, but computer science in Waterloo was under mathematics) the first question my mother asked me was, "When are you going to be an engineer?" Well I'm a little beyond that now, but no I'm not planning to become an engineer. [Laughs] I'm not going to build any bridges or any machinery. So my plan was to do engineering. I didn't do engineering in Uruguay. I ended up doing computer science, which was much easier for me; it was actually a

little bit shorter as a degree. After finishing the degree in computer science, I went to study abroad.

HAIGH: So was it officially a degree in "computer science" at that point?

GONNET: Yes, it was a degree in computer science. It was a university degree in computer science, which I had studied at the same time that I was teaching, so for some courses I would teach the courses and some of the courses I would be the student.

HAIGH: Was this a degree that was offered within the mathematics department or was it separate?

GONNET: Actually it was an independent school inside the University, because the University did not know where to put computer science. And in not knowing where to put computer science, it was physically housed at the engineering faculty, actually in the Mathematics Institute. It was sort of a single branch stemming from the top of the University. Later it was incorporated into engineering, as is the choice in many places. But originally it was sort of an independent degree. So as a matter of fact, my original degree is not a degree in engineering (to the chagrin of my mother); it was just a degree in computer science.

HAIGH: Now, did your interest in computer science develop from the work that you had been doing at IBM or was it the other way around?

GONNET: It's difficult to say—chicken and egg type of thing. I think that I was interested in mathematics, and I had to earn a living, and so working with computers, or working with unit record equipment and cards, was a way of making a living. Then all of a sudden, when it comes to scientific computation, you see that there is more to programming than just boring payroll programs, that there is more science to it. Then the purpose of earning a salary goes to the backburner, and this becomes an interesting academic activity in itself.

HAIGH: So the initial work you had been doing for IBM had been on these kinds of administrative and routine applications.

GONNET: Right! Very much routine.

HAIGH: Were those well developed in Uruguay at this point, or was it a novelty?

GONNET: Computers were a novelty. I'm talking about early 1960s, so not everybody had a computer. The applications were also very trivial. Most of the applications were card-oriented. People would not trust their data being stored in magnetic ways at the time. They would prefer something as solid as a punched card. I wouldn't say that Uruguay was particularly less developed than other countries or more developed than other countries. I think it was following the trend of Latin America at the time. We were not really in terrible shape. When the University finally bought their first major computer, which was an IBM 360 model 44, at the time that machine rolled into our building, it was the most powerful machine in South America. Now of course, soon after, somebody bought a bigger machine, but that is to give you an idea that we were not the bottom of the stack; we were doing sort of average compared to the rest.

HAIGH: Before you got the job with IBM, did you have a clear idea of what a computer was?

GONNET: No. Actually I learned what a computer was at IBM. For quite some time, living inside IBM you get a very skewed view of the world. The world is cards of 80 characters and computers that work in this way and so on. The 360 was the computer that appeared at the time that I started computing. So it was sort of the fashionable thing to do at the time.

HAIGH: Can you remember anything about your personal reaction to computing? Was it something that you became enthusiastic about? Did you enjoy programming?

GONNET: Yes, I definitely enjoyed programming. I remember probably a couple of anecdotes that are worth mentioning. The compilers for the 1401 were extremely primitive. We're really talking about very basic assemblers, and there were basically two assemblers to speak of at the time that we were working on these machines. Autocoder was more modern but consumed a few more resources, a little bit slower to compile and so on. Management was adamant: "No, no, no the machine time is too expensive. You should use the more primitive assembler, SPS." That was just an assembler with a few features like free format and literal constants and so on. You could say, "Add one," and just put the one in SPS. To add one to something you had to define the constant one as a constant in someplace, give it a name, and then say, "Add this here." It was just an extremely fixed and poor assembler. Something like the op code has to be in columns ten to twelve; the first argument in columns whatever, and so on. Not even literals, no constants, everything had to be defined and done by your own standards.

And I was trying to push for a better programming language, saying we are going to code better programs and we are going to be more efficient. "Oh no, no, no. It's not efficient programmers what counts; it's how fast the computer is going to run." I guess that was the view that prevailed at the time, that computers were indeed very expensive, so computers had to be maintained and run as long as you could. And programmer's time was not that expensive at the time, and the programs were also much simpler.

I also remember that at some other point I was also involved in buying another computer for another organization, and as a requirement we said it has to compile COBOL, and this outraged IBM because what a ridiculous thing. "What is Gaston thinking? This is a ridiculous requirement!" And that was because they were forced to bid for a higher cost computer and they wanted to bid for a lower cost computer that could not run COBOL. But some people viewed the idea of using a high level programming language as an outrageous waste of resources. Quite a different view from nowadays, of course.

Even the COBOL that we were talking about was still a very primitive language compared to anything that you would want to use nowadays. Yet, and this is an interesting coincidence, at the time I developed a little program to do symbolic derivation, and very quickly I realized that doing symbolic derivation was very easy. The problem was simplifying the expression that was coming back. There was another thing that I had to work at the time, which was one of the distinguishing features of Maple. (In Maple it is nowadays called Option Remember). It's the ability to remember the results of a function evaluation in one argument once you have performed an operation. In the case of computing derivatives this is very important, because the computation of derivatives is very repetitive if you have a large expression. You see the same set expression again and again and again, and so you would save a lot of effort if you reuse what you have already computed. I can say, in the late '60s or early '70s, I was already doing a little bit of symbolic algebra and I was already using Option Remember which is a feature that appeared years later in Maple.

HAIGH: On your resume it says that you were from 1969 to 1974 working in the University Computing Center.

GONNET: That's correct. The 360 machine at the University arrived in 1969. The Computing Center and the Computer Science Department were one. This was the Scientific Computing

Center. It was not the payroll or administrative computing center; it was the Scientific Computing Center. But both the academic part and the computational part were done by the same people and in the same building and so on, which is not a bad idea because there were many users from physics that were doing very interesting work, people from engineering also doing very interesting work, and so it was a challenge to understand what they wanted to do and try to help them and basically work with them.

HAIGH: How would this scientific computing center have been organized? Did it have a large staff attached to it?

GONNET: No, it was a relatively small organization. There was a director who had turned out to be a very good friend of mine in the end, Dr. Luis Osin who had a Masters in Computer Science from MIT, and when he went back from MIT to Uruguay he started organizing all the academic part. I was part of the teaching/academic staff. There were some regular employees that were more programmers than anything else, and there were some people that were actually operating the machine—these were computers that required an operator.

HAIGH: So there was a specialized operation staff, it wasn't that you could sign up for time on the machine and actually work it yourself?

GONNET: The inner circle, yes, but not the rest of the students. This academic computer at the time had as its only input punched cards. It had a console, but the console was deemed to be too precious for anybody to type on. And disk space was also at a premium. It had two removable disks of one megabyte each—one megabyte each!—in its original form. For machines at the time, it had a very large memory, had 128K. And it was quite a fast scientific processor, so that was a step forward for scientific computing in the country. I once made a comparison with the laptop that I just recently stopped using, and it was very nice to see that there was a factor of 1,000 on everything. My old laptop had 128MB, the IBM had 128K, it was a factor of 1,000. The speed, there was no question there, was a factor of 1,000 faster. The software is probably a factor of 1,000 better than what it was before. And the original machine was $400,000 US dollars. I'm talking about US dollars in 1968. That was a factor of 1,000 times more than what my laptop cost me. There's also no question that the weight of the two machines was also separated by a factor of 1,000. So you had a factor of 1,000 on everything with respect to that machine, and by now it would be even bigger, right? This is four or five years ago that I made these computations.

So it was a good start. It was a good environment. Primitive in the terms of what we may do nowadays, but enough to develop the academic part, and to a certain level the scientific part, the scientific work that was done at the University. It was actually a very pleasant surprise when I went to Waterloo to find that they had a better machine. They had a 360 model 75, which was their biggest machine, but side-by-side to the 75 they had a 360-44, and this machine was not half but a quarter of the computing power of Waterloo at the time.

HAIGH: Would scientific users have written their own problems? Or was there a team at the computing center that would write the problems for them?

GONNET: No, scientific users would write their own programs. Fortran was the language used for that computer all the time. Assembler was not suitable, and Fortran was very efficient to compile and to execute in that machine. All applications that I can think of were written in Fortran. So Fortran was the *lingua franca* for that stage of computing.

HAIGH: Can you remember what the main kinds of problems running on the machine would have been?

GONNET: As I said before, some people in physics were doing some semiconductor experiments. I know a little bit more about what they were doing because I got involved at some point. They were basically doing differential equations, solving relatively simple differential equations to study how semiconductors would behave. That was one application that was very significant. Other people in physics were doing more standard work, which probably most of the time boils down to solving differential equations. Some people in mathematics, in particular in statistics, were sort of discovering singular value decomposition at the time. Singular value decomposition is something that you don't do by hand. It's too computing-intensive, so they were very happy to be able to find eigenvalues/eigenvectors of large matrices automatically, which, again, is something that you definitely don't do by hand.

We had a lot of ad-hoc little things that were done—some small, some big. At one point we processed the national census. This was the biggest computer in the country at the time, and so it was deemed the most suitable computer to use for the census. That ended up being a huge job for us because of the memory restrictions, with only two megabytes of magnetic disk to use. So we had to stand on our heads to read all the data only once, because the only input that we had was punched cards. Although the country had only three million people, I think it was a little bit more than three million cards. But three million cards, no matter how fast you read them, takes a very long time to read them all. So definitely wanted to read the data only once, and that forced us, a little bit, to stand on our heads.

HAIGH: Did the machine have tape drives?

GONNET: No, no tape drives. That machine was then extended to have more disks, 30-megabyte disks, but for whatever the reason, it never had magnetic tapes. I guess there were reasons of cost in the end. Computing equipment was very expensive, and the university did not have tremendous amounts of resources. They had invested, for the university and for the time, a huge amount of money in this computer; it was not going to be easy to extend it.

HAIGH: What work were you personally doing in the computing center after the 360 arrived?

GONNET: Well, I was involved with the teaching. It was definitely a large involvement. I was involved with the operating system, trying to make it a bit more efficient, a bit more pleasant, changing some things here and there. It was a time that you could do this; the operating systems were simple enough that with some amount of effort, you could master everything that was happening.

HAIGH: Was it running OS/360?

GONNET: No. The Model 44 had a special operating system, which was simplified operating system. I don't remember how it was called, but it was a special operating system for the Model 44. This was because the Model 44 had a restricted set of instructions. Basically the memory-to-memory instructions that are present in all 360s were not present in that machine. This was, in the end, a marketing decision. IBM did not want to release a machine that was so powerful CPU-wise and useable in business because they were charging a lot more money for the commercial machines. Yet they wanted to cover the academic market in some way. So the way that they approached it was to design a CPU that had all the memory-to-register operations; all the floating point operations are very efficient, memory was efficient, but would not have the ability to compare memory-to-memory and so on. And so it had a special operating system and it has special Fortran compiler, and it didn't have a COBOL compiler, for example.

HAIGH: So you were rewriting some of the code in the IBM-supplied operating system?

GONNET: Yes. Well, I remember developing an on-the-fly linker, because the standard operating system was compile, link, and then execute. When you run student jobs and you want to have the highest possible throughput, every 10% that you cut or every 20% that you cut of execution time is really important. So I remember developing this on-the-fly linker, so that it would compile and execute right away. A not insignificant part was that at the time, everything would be printed on pages. The linker would typically print two or three pages, so every little job had two or three extra pages wasted. We were very sensitive to that waste, so we wanted to eliminate that step.

HAIGH: Were you ever personally involved in working with users to help them solve particular problems?

GONNET: I guess that that was a natural assumption. Everybody that was working there on the more academic side would be cooperating with people that had programming problems or numerical problems or whatever.

HAIGH: Do you think there's anything from these years that you spent in the computing center, with at least some exposure to the kinds of problems that people were trying to solve, that influenced your later career in terms of working on software that people would actually be using to do things with?

GONNET: No, I wouldn't pinpoint anything specifically, other than the fact that you have to be a jack-of-all-trades because we were just very few around. So other than that…no, I wouldn't say that there was anything specific about it. There was definitely a mathematical orientation to the whole institute, to the whole computing center. I was personally involved with kernels and operating systems, which were called supervisors at the time. But I wouldn't say that there was anything that said, "Oh yes, there was some important event that marked the rest of my life."

HAIGH: All right. Just before we move on to pick up in Waterloo… We've covered the Computing Center. Now moving back to your experience as an undergraduate, I was wondering, were there any areas that you were exposed to during your time at university in Uruguay that turned out to be particularly important later in your career in terms of intellectual development, or any relationship with colleagues or faculty or other students?

GONNET: Nothing in particular, I would say. The general inclination that the whole environment had towards mathematics helped me tremendously later. Some colleagues that I had that kept me honest with my mathematics…or kept me honest with my science, in some sense. "No, you don't do that that way. That's wrong. Do it right. You know how to do it right!" Living in such an environment is sometimes very helpful. In general, in computer science, I have always been treated more like a mathematician, whereas my mathematics colleagues think of me just as a programmer, maybe, and not very mathematically inclined. So I make everybody unhappy in the end.

HAIGH: Your résumé lists you as Assistant Professor, Second Degree, from 1966. So you were working for the University before you were a student?

GONNET: That's correct.

HAIGH: You were also teaching courses while you were an undergraduate. Was that unusual?

GONNET: It had to be done, somehow. I was teaching more courses related to operating systems, and I would say that was my area of expertise at the time. And hardware, too, I understood slightly better than the others how the hardware was working at the time, so I could

teach some courses in hardware. I think I was also teaching courses in COBOL because of business needs.

HAIGH: You mentioned on your resume that you had served as a computer advisor to a health insurance firm.

GONNET: This was a peculiar system, almost like a cooperative, a very large institution that was providing healthcare privately to citizens, but was organized by the doctors' associations somehow. There were several such institutions, but this one was the largest one. And we started consulting as part of the University and then became consultants and developed their computing quite substantially. And that gave me, well, it was a job. I was earning some revenue, some money. But it also gave me quite a bit of a sense of the real world computing because people needed to do payroll, they needed to bill customers and so on. There was a need to design systems that will work with minimal user interaction and flexibility and serving customers adequately and so on.

HAIGH: Do you think there was anything in that experience that's influenced your life and career?

GONNET: No, I wouldn't say so.

[Tape 1 of 6, Side B]

HAIGH: So you've already told the story of how it came to be Waterloo that you applied to. Now at that point, would it have been feasible to think about staying in Uruguay or elsewhere in South America for your Ph.D.?

GONNET: No, no, certainly not for a Ph.D. We didn't even have a Ph.D program at that time. Even if there was one, the country needed desperately to farm people abroad to bring back technology, know-how, ways of doing things and so on. So many people recognized that it was very important to have at least one generation of teachers that would be educated abroad so that they could being technology, could bring know-how, could bring whatever connections, just if you wish. So it was very clear, in my mind at least, that I had to go.

I have to say that the political system helped me leave very decisively. In 1974, Uruguay was unfortunate enough to go into a military dictatorship. And the military dictatorship devastated the university and the government found no better thing to do than to chase all the professors and the researchers away and "punish" them for being independent thinkers—whether they were in favor or against, it didn't matter. So the university suffered tremendously. When this was starting to happen, it was the time that I left. I had already planned to leave so it was not that I was leaving because of this; but if I had any hesitation about leaving or not leaving, they made my hesitation disappear very quickly.

So in some sense, it also helped me make the decision because once you have decided to go, it's a big step. Nowadays it's easier to go and study abroad. In the early 1970s it was an adventure to go half-way across the world to a place where you don't speak the language very well or that has a completely different way of life. I remember something very funny. I had always lived in a big city. Montevideo is a million and a half people, and it is very cosmopolitan. So I looked at the map and I saw Waterloo, 50,000 people—I was really worried that I was going to go into a hole in the ground. This was a serious concern, believe it or not. Will I have a movie theater when I wanted to go? Will I have whatever I need from a big city? I was really worried, and I could only put my worries away once that I learned that Toronto had about two million people and was only

100 kilometers away. I said okay, if we lack anything, it's only 100 kilometers away that we are going to be able to find everything that we need.

It turned out that then we seldom went to Toronto for anything. Waterloo was a very nice place, and actually there was something misleading. Kitchener and Waterloo and twin cities, and although Waterloo was very small, maybe 40,000 people when we arrived, Kitchener was much larger, and the two together were above 100,000. In a city of that size in North America you normally find everything that you want. I wouldn't say that it was the same as living in Toronto for the cultural aspects, not even remotely, but at least it satisfied all my needs and I never had to worry. But from the distance, you look at this place and you have a very distorted view of how it will be.

So it was a little bit of an adventure to go there. And definitely, at the end, the political situation pushed me to go because I was going to end up chased around, manipulated—a very unpleasant situation. So in some sense, that signified also a final separation for me from Uruguay, because when I had finished my Ph.D. I could not return to Uruguay. The situation had worsened with the military dictatorship, and there was just no chance of returning. So I went to Brazil. Finally, when it was possible to return to Uruguay, which was in 1984 when the dictatorship finally ran completely out of steam and the country returned to democracy, it was too late. I had been away for too long. I just could not see myself going back to the things I was doing. It would have been too much of a loss in my career, and I would have not been used positively by the country. In some sense I went out of phase with the country. For some time, you can reintegrate yourself. At some point, it's just not possible.

HAIGH: When you turned up in Waterloo ready to begin your studies, was that your first visit to North America?

GONNET: No, I had been in the U.S. as an exchange student when I was a young teenager, so I knew a little bit of American life. I had been in the closest thing to a hole in the ground, which is Coldwater, Michigan, near Ann Arbor. Curiously enough, I ended up in Waterloo, which is only about 300 miles from Coldwater. From that point of view, no, it was not a shock. I sort of knew what was happening. I knew what a cold winter was. It was a very good time. I integrated myself into the university straight away. I found excellent people around. I was taking courses that I really enjoyed. I have to say that I had a great time in Waterloo. I had lots of teachers that were exceptionally good. Not only technically, but also humanly. So it was a very pleasant time.

It was a time of very high pressure for us. From the economical point of view, we didn't have any resources that we could bring from Uruguay. I didn't have a salary or a grant or anything because of the situation, in particular because of the military dictatorship. So I was really on my own. The university and the professors were very cooperative, and they helped me tremendously by giving me research assistantships and so on, but it's very difficult to make a living out of those. I was married already. And so it was basically a race against time. We had some savings, and every month the savings were going down, and I knew that when I was getting to a level that was the value of two tickets to go back, I had to leave, whether I had a Ph.D. or not. Still it was a very good time in terms of academics and in our lives. We had our first child. Actually, we had two children while we were students in Canada. My daughter was born the day after I got my Ph.D., but my first son was born in Canada before that. So I did a master and a Ph.D. My wife did a degree in computer science, all of that in scarcely more than two years. I actually had to set a record in speed of Ph.D. because I couldn't afford more time.

HAIGH: When you went to Waterloo, had you already known that you wanted definitely to stay for a Ph.D.?

GONNET: Yes. Definitely my idea was to go for a Ph.D., but neither Waterloo or Essex would accept me for a Ph.D.. They said, “Well you come here for a Master’s.” And I was happy about that. It’s understandable. The same things were happening in reverse. These people were looking at me. Uruguay? They would probably have to look at a map to find where it was. “Do these people have a university? Do these people have streets? Do they live in trees? Do they live in houses?” So I was one-of-a kind in some sense. And of course, once that one goes and they see that, oh, “They sort of have two arms and two legs and a head and so on,” they appear to learn. Then you open the door for other people to follow the same path. But in many ways, for Waterloo, I think I was certainly the first from Uruguay, and most likely one of the first students from South America. So it’s perfectly understandable that they would accept me only to do a master’s, but that was never a problem. I was a reasonable student so I didn’t have any problems in my academic life.

Actually most professors were very surprised that I wanted to leave so quickly. “Are you not enjoying your time here?” “No, I am really enjoying it, but I will run out of money, and I will have to go.” So I don’t regret it at all. I think that just at the time the Canadian immigration policies had been changed and tightened up tremendously, so it was very difficult for my wife to get work and to help me. And I was very restricted on the things that I could do. So that was unfortunate. I didn’t have any time to relax, but that actually was good in the end. I am very happy about my life as a student in Waterloo. It was a very pleasant time. Stressful, but pleasant. It’s difficult to believe it, but it’s possible.

HAIGH: How would you, in general terms describe the state of development of the Waterloo computer science program at the time you arrived?

GONNET: Waterloo, at the time, was in transition in many ways. The department was very visible in the Canadian scene, for sure, but in North America as a whole was one of the best departments doing scientific computations, which at time was called numerical analysis. Actually the department was called the Department of Applied Analysis and Computer Science, which was sort of a special name. Then it changed to be just Computer Science, but at the time it was called Department of Applied Analysis and Computer Science.

HAIGH: Was that a separate department from the Mathematics Department?

GONNET: No, actually it was a very peculiar structure. There was a Faculty of Mathematics, which is an oddity, and the faculty had five departments: Statistics, Pure Math, Combinatorics and Optimization (a very well known department), Applied Math, and Computer Science. So there were five departments, and we had our own building and so on. That actually was a very nice set up for me, in particular, being mathematically inclined because even though we were in the department of computer science, we were not in engineering or art or in some other place. We were in the faculty of mathematics. And my degree is a Ph.D. in Mathematics. Computer science, but in the Faculty of Mathematics.

HAIGH: As you studied there initially in the masters program, were there any areas of study that you were particularly attracted to or faculty members who you feel had a lasting effect on your development?

GONNET: Definitely. As I said, I took lots of courses. I wanted to absorb as much as possible. I had this mentality that I was there to absorb as much as possible to bring back to my own

institution, so I was enjoying every course and absorbing as much as possible, taking notes, trying to understand everything. My supervisor for my master's thesis was a professor by the name Lawrence "Laurie" Rogers. He was just a superb guy, and we got along very well. I liked him instantly. And he taught me about algorithms and analysis of algorithms. It was really him who put me into a track of analysis of algorithms. And because I had a good mathematical background, in the end it was easy for me to do analysis of algorithms. We got immediately into an excellent relation there, both academically and also as a personal friend because Laurie Rogers is just a great person. It was really a pity that he moved to California. Well, not a pity, he had to do it; but for me it was a bit of a pity. Laurie Rogers was a very good friend of Alan George, and Alan George was doing scientific computation and was also very mathematically inclined, and so Alan George inherited me from Laurie Rogers when he left for California.

HAIGH: So George was your dissertation supervisor?

GONNET: That's correct. And my Ph.D. thesis was on analysis of algorithms, mainline analysis of algorithms as a matter of fact. Both my master's thesis and my Ph.D. thesis were on analysis of algorithms.

HAIGH: Were there any other faculty members or students that you had an important relationship with?

GONNET: I had written a couple of papers by then with some other professors, but I wouldn't say that there was anybody there that had influenced me especially. I published lots of papers together with Frank Tompa, and we became excellent friends, but actually the relationship with him probably did not start until I went back to Waterloo as a professor. Although we knew each other and I had worked with him, we didn't have a close academic relationship before that. I was there for only two years and a term for a master's and a Ph.D., so it was really a bit of a rush altogether.

HAIGH: You said your Ph.D. [Interpolation and Interpolation Hash Searching] was fairly mainstream work. Was there material in it that you later picked up and published and developed further, or was it just something you did because it was easily doable in the timeframe?

GONNET: It was analysis of algorithms, and this was the analysis of what is called linear probing search. It's a technique for searching that had surprising behavior. It's the type of results that you find, "Okay, this is the result, it was a log-log behavior." There's not much more to say. So it's not the type of work that you can make a career out of it. It's, "Here is a result. Nobody knew this before. Now, everybody knows it." There's not much more to say.

So from that point of view, the Ph.D. produced one good paper and I think another note or something like that, but it didn't start a career. Analysis of algorithms is a little bit like that. You work in this algorithm, you prove something or you resolve it completely, and you go to the next. It's like processing sausage: you make one sausage, it's done, consumed; next one, done, consumed; next one, and so on. So I wouldn't say that the Ph.D. topic had any lasting impact or any other consequence.

HAIGH: As you approached graduation, how did you begin to plan for the future?

GONNET: There was actually little planning. I was worried about where I would go after graduating because of the situation in Uruguay was not getting any better; on the contrary, it was getting worse. All of the sudden somebody told me there is an opening in Brazil and you should be interested. This was professor Don Cowan, who had some contacts by that time with some

people in Brazil in Rio de Janeiro. All of a sudden, next thing that I know, I have an offer in my mailbox to go and teach in Rio de Janeiro. We had been as tourists before in Brazil and we really liked it and I thought it was a good alternative to spend some time in Brazil in what is a very nice city from the geographical point of view. I often say that I have never applied for a job, and I have never gone to interviews for a job. And it's still true. So all of a sudden I had this offer, I accepted it, and I found myself a few months later in Brazil.

Brazil was not a great experience for us for various reasons. The country there is so beautiful because of its geography and because of its beaches and its location and everything. It's not the same when you have to live in there every day. There is a marked difference in social classes. You realize that if you are going to live there you are going to lose part of your culture. You have to play the game. You have to live in the place, you have to accept the whole package. We also had bad luck, we lost our first son to an accident. I also got seriously ill and it was not a very pleasant time, although it is not necessarily the fault of the country.

HAIGH: So how was the university itself?

GONNET: The university was a very reasonable place within the context of the entire country, probably the best university in Brazil from the computer science point of view. It was called the PUC, Pontifícia Universidade Católica. I am completely non-religious but I ended up teaching in it. It is basically a private university where the high class kids go, and it was reasonably well funded at the time and in a very privileged part of the city, and so to be working in that particularly place was a little bit of a ghetto inside the reality of Brazil. But my colleagues, most of them or all of them, had Ph.D.s done abroad, so it was a very good academic level. As I said, my complaints about Brazil have nothing to do with the university; they have more to do with our personal situation at the time. When you leave your home place, you either leave it because you are going to go into an academic place that is really challenging, or you going to make lots of money, or you are going to live in a fantastic place, or you are going to be very close to home. And Rio de Janeiro was neither of those. Actually it was getting close to most of those, but it was none of those exactly. So we were neither here nor there. So after we lost our son, I received an offer from Toronto to go back as a professor, and almost immediately I prompted an offer from Waterloo because somebody called me and I said, "I have an offer to go to Toronto. Are you going to do anything?" [Chuckles] Soon I had two offers, one from Toronto and one from Waterloo, and I decided very quickly to go back to North America, as a matter of fact.

HAIGH: And why did you choose Waterloo over Toronto?

GONNET: Some people would say to annoy the people in Toronto. I don't know. You may not be aware of this, but it has always been the view that Toronto is *the* University in Ontario, in particular in computer science. They have been looking down at Waterloo all the time because they do have their great people. So when they made me an offer and I accepted an offer from Waterloo instead, they were not happy at all. I think they have never forgiven me for that.

But all in all, we knew Waterloo; I had very good contacts. We had friends. The environment is important for what you do, and a positive environment may help you. But the difference between Toronto and Waterloo, for me at least, was not going to be of any substance. If I was going to do something, I was going to do it in Waterloo or in Toronto—the means were all there. Maybe if we had said the University of Manitoba compared to Stanford, yes there is a substantial difference in what you may do in some place you may not be able to do in the other. But this was not the case between Toronto and Waterloo. I voted basically with my heart and my friends and

the people that I knew that were extremely reasonable. So I didn't see any way of hindering my career. Neither when I went to study, nor at the time that I went back as a professor, was I ready to go to the States and live in the States. So the States were basically ruled out.

HAIGH: And why was that?

GONNET: Well, having lived through a military dictatorship that was sponsored by the Americans, you end up not being very friendly, or at least not wanting to pay taxes to the same system. So yes, the military dictatorships in Latin America of the time were all born or created or manufactured or fostered or financed by the U.S. We are not very happy about that. We are not going to forgive that. It costed many lives. It costed many years of pain to the country. It is not something that is easily forgettable. The Americans will have to live with it. And there is a lot more to come, let me assure you.

HAIGH: So did you find that it was a different experience at Waterloo as a faculty member than as a graduate student, or did it feel familiar?

GONNET: It felt like going back home in some sense. I felt right at home from the first day. So I knew where to get things, I knew my way around, I knew the people. I hit the ground running, you can say.

HAIGH: So can you talk in general terms about your first few years there as assistant professor. Perhaps experiences, relations with colleagues, important papers or areas you began to work on?

GONNET: It was again a very pleasant time. I was doing analysis of algorithms, and at that time I was working mostly with two colleagues: the one that I mentioned before, Frank Tompa, and Ian Munro. Instead of working in the general algorithms area, applied algorithms, text searching, and so on, whenever we were publishing a paper together, I was the one that had to do the mathematical analysis. The other ones were putting in the brilliant ideas; I was just putting the donkeywork underneath. And in fact I was a consumer of computer algebra. And at the time the situation for getting computer algebra was very unpleasant. We had to either login remotely on various phone lines that were being charged long distance to MIT where we could get a free account and run things on MACSYMA, or we could run an algebra manipulator called ALTRAN, which I probably still have a book over there, which was a batch-oriented type of system, very much like Fortran. It was a bit of a disaster, but was the only thing that was around. At the time Waterloo had a very nice time-sharing system. I'm talking about the late 1970s. It had a time-sharing system based on Honeywell equipment. It was very responsive.

HAIGH: Would that be MULTICS?

GONNET: No, it was not MULTICS. It was running on very similar hardware to MULTICS. MULTICS was what was being run at MIT. But this was not MULTICS. I don't remember what it was called, to be honest.

HAIGH: I see. Was the Honeywell operating system GCOS.

GONNET: I think that's right. Yes, that was GCOS.

They gave us a very small environment. Well, small for nowadays of course, but even small for the time. It was a very fast interactive type of system, and we always said what we really need is an algebra system that is somewhat smart but runs on the machine that we interact every day. So it was very clear in our minds that it was much better to have something simple, not too sophisticated that we could run on our desks. We turn around, we type something, and we are

running on it, as opposed to this sending a job to ALTRAN that is in the queue for half an hour and then gets executed in the next two hours and then it comes back and it has a syntax error on the first column and it's all wrong. Or else logging into MIT, which is running at a dollar a minute at the time or even higher, so by the time that you have typed a formula and got an error message, you have spent a huge amount of money.

So in 1980, prompted by an article that appeared saying that what we need in computer algebra is for Waterloo to produce something equivalent to WATFOR for computer algebra. I think I can dig the reference to that article. A group of us got together to basically think about whether we could build a computer algebra system or not, or what could we do with computer algebra at Waterloo. I remember exactly who were the people meeting there: Morven Gentleman was sort of the computer scientist with a tradition in numerical analysis who had turned into more operating system design and so on; Mike Malcolm, who again was scientific computation, Stanford graduate in numerical analysis; I think Gene Golub and Forsythe's student; Keith Geddes (who is going to be named many more times), who was also a numerical analyst inclined to computer algebra; Wesley Graham (the father of WATFOR and WATFIV); and myself. The goals of the participants were very different. Keith Geddes wanted to buy a VAX machine so that we could run VAXIMA, which was a version of MACSYMA implemented on the VAX. Wes Graham was obviously of the theory that we should develop our own software. I had developed a very small system, as I said before, which was what I wanted, called WAMA, "Waterloo Algebraic Manipulator", which was just a modest set of functions that was completely interactive and would run on this time sharing system on the Honeywell. For me it was already more useful than the bigger systems, because it was doing something simple and I could do it interactively. The power of interactivity was very obvious, at least for computer algebra. Mike Malcolm, who was interested in systems—was also working on operating systems and systems design at the time—he was more inclined, together with Morven Gentleman, in looking at whether or not we could do something from the systems point of view.

The result of that was that, I may say that I got my way there, or Wes Graham, got his way, except that Wes Graham and Mike Malcolm didn't get very much interested in this product after. It was not what they wanted to do, so they didn't meet any further. Morven Gentleman, Keith Geddes, and myself were definitely the main people involved in this, and Morven Gentleman was very encouraging for quite some time. Then he became also a little uninterested when we became too much computer algebra oriented.

HAIGH: What did he have in mind as an alternative to being computer algebra oriented?

GONNET: I think that Morven Gentleman was not a big consumer of computer algebra. He was interested in the system design. He gave us very good input. It was critical at many points and he told us very important things. But I think that it was not his baby or it was not his main interest. He was actually not doing scientific computation any longer, so yes he was interested, he was helpful and so on, but he was not going to be a main player. Although he was for a very long time very much active and participating in giving us ideas and even testing the system and so on. I think that if we go to the early mail we should see Gentlemen copied in most of the e-mail.

I had a very practical approach. I had started this WAMA, Waterloo Algebraic Manipulator, and I was not the type of person to say fine, let's do a committee and let's apply for a grant and two years from now we are still discussing. In a few weeks I had something running. And I guess that was key to the whole project, that very quickly we were testing and running and improving the system. I have to say that at that point Alan George also played an important role. He had

become dean of the math faculty, and he perceived that we had something there that was worthwhile. And so before we could even get our act together and our papers together and everything, he managed to get us some sort of a starter grant that allowed us to buy some equipment and hire some people, students, to start working on Maple.

HAIGH: Do you know when that grant would have been received?

GONNET: Possibly it was in 1981. It was very early. The date of the first compilation of Maple is December 8th of 1980. So Maple didn't exist before that time.

HAIGH: Can you remember where the name came from?

GONNET: Yes, yes I remember quite well where the name came from. We were discussing the name, and the mathematic programming language sort of was there. Keith came up with some acronyms that sounded like "Maple," but were not exactly Maple and were really acronyms for "mathematical program language" with some other additives. And at the end I said it should be Maple because it is a good Canadian name and it should reflect that, and whether it has a acronym or not, it doesn't matter. So it was a joint decision. I would credit Keith more than anybody else with coming with some approximation of Maple and then me saying, okay let's call it plain Maple and that's it. Actually the name has been a good name all along. I think we have never regretted the name.

HAIGH: So it never officially stood for anything?

GONNET: I guess that is true, although we say at some point that it stands for something.

HAIGH: Even on this early paper it just says "it is not an acronym but rather simply a name with a Canadian identity." [Bruce Char, Keith O. Geddes, W. Morven Gentleman, Gaston H. Gonnet, "The Design of Maple: A Compact, Portable, and Powerful Computer Algebra System," in *Lecture Notes in Computer Science 162: Computer Algebra*, edited by J.A. van Hulzen, Springer-Verlag, 1983, 101-115, page 102].

GONNET: That is very accurate.

HAIGH: Okay, so we should talk in a minute about exactly what the initial version was and how it developed as you got the grant money. I wonder if you can say just a little bit more before we do that about how it was intended to be different from the systems that already existed. I know IBM had the SCRATCHPAD system.

GONNET: No the SCRATCHPAD is later as a matter of fact. We are talking about 1980…

HAIGH: I know it was only commercialized as AXIOM much later, but I found a reference that suggested that the SCRATCHPAD work had been underway since 1971 with IBM in some form.

GONNET: Oh well okay, yes. I cannot challenge that, but I would not say that it was a player at the time at all. At the time there were only three players in the computer algebra community. It was MACSYMA, which was the most sophisticated system, in the sense that it had the most and best algorithms and the most mathematics in it. Then it was Reduce, which was used extensively by physicists, because it was designed by a physicist. It was very competent in certain computation. Then there was ALTRAN, which was sort of this Fortran preprocessor type of batch language. Then there had been some others that were not really options if you wanted to do computer algebra.

HAIGH: And where were Reduce and ALTRAN coming form?

GONNET: Reduce was the product of Tony Hearn and was maintained by him for a long time, with many other colleagues, but I think that he was the main force behind it. ALTRAN was really a product of Bell Labs, or AT&T or whatever it was called at the time. As a matter of fact Morven Gentleman had some experience with ALTRAN so there was a connection there. And of course, Keith Geddes had some connection with MACSYMA. I don't remember exactly how it was. Now remember that I was not a computer algebra person. I was an algorithms guy. I was coming into computer algebra as a consumer. I didn't want to design computer algebra algorithms, I didn't want to do anything in computer algebra, I just wanted to use it. Now MACSYMA was the one that would come closest to an interactive system that was usable, if you have a hugely expensive MULTICS machine behind you. Reduce was sort of interactive. Both MACSYMA and Reduce were based on LISP, and that made them very clumsy, very large, very slow to operate and so on. So some of the things that we had in mind from the beginning were: we are not going to use LISP because LISP means a huge baggage of things that you are going to regret sooner or later. LISP would mean that we don't have a machine where we can run it. LISP will mean speeds that we are not going to reach. LISP means a very ugly language to code in. Even though some people love it, it is really very ugly as a modern programming language.

[Tape 2 of 6, Side A]

GONNET: I was saying that Morven Gentleman had a lot to say about programming languages. This is a time when C didn't exist yet. An ancestor of C existed, which was called B. Now B really was coming from BCPL, which was a language that was developed in the UK at some point, and was a very peculiar language. A derivative of BCPL was developed at Bell Labs and was called B, and was developed on this Honeywell computer. Actually, the main language for developing software on the Honeywell machines was B. Out of B came C. It was a funny thing about Kernigan being asked whether the next language was going to be D or P, and he says, "No, I'm never going to restrict myself to four languages," so he was going, you know, B, C, P, L. The truth is he never went beyond C. In any case, the language that was chosen at the time was B because it was definitely a model language at the time. It was a language that would compile very efficiently, and it was a language that people believed was going to be portable to many other computers in future. At least, Morven Gentleman was of that idea, and I think that proved to be roughly correct. B was, in the end, very close to C, and C became widely portable. But you have to remember that these were the times before Unix. Unix was something that Bell Labs had not disclosed yet, or was still definitely not available outside of Bell Labs.

HAIGH: And, in fact, portability is mentioned as one of the goals in one of your 1983 paper.

GONNET: Right. Portability was one of the goals. But I would say that having an efficient language, having a small system, having an interactive system were the dominant goals at the time. Of course we wanted it to be portable, but I think that small, efficient, and interactive were the keywords, because we had experienced the pain of systems that were maybe more sophisticated but were neither of those, and that was clearly not the way to go.

HAIGH: Maple was a computer algebra system and a programming language, and as you've mentioned, your main previous area of research wasn't any of those things. What do you think that you brought from your existing knowledge and experience that particularly shaped the project?

GONNET: In retrospect, 25 years later, what did I bring to the project? I think I brought energy, because I was not a computer algebra person. I was going to be a consumer of computer algebra,

so in my ideal situation, I should have gone to these people and said, “Give me a computer algebra system. I’m going to use it.” Well, they didn’t have a system to give me, so I developed one. I had always been a reasonable programmer. Maybe some of my students disagree, but I think that I could produce code. I enjoyed actually producing code, and I was more, “Okay, let’s get it done,” you know? “Let’s stop discussing about it and let’s get it done now. Once we have a version we can discuss it again.” I think that that was something that I brought into the group.

We mentioned this before; I would say it again for the record. I think that successful projects are identified by having consumers. Consumers, in that academic sense, could be anybody who is interested in your results. Whether it’s a theorem, whether it’s a theory, whether it’s a program, whether it’s a corollary—if people are going to use or are going to consume what you are producing, then your project is very likely to be successful. And I think that I was the first consumer of Maple, and I think that Maple very quickly had lots of consumers. Maple did not develop to what it developed because there was a group of computer Algebra people interested in doing such a thing in Waterloo. Maple developed because we wanted to solve problems every day. Now of course, the scope of the problems changed with time, and I became a computer algebra person. I published papers in computer algebra in the end. So the means became the goal for me at some point.

But what I think I brought to the project, I would say it was the programming power of being able to say, “Okay, fine, we are going to do it, and here’s the first version.” If you are going to plan forever, forget it. “Here’s version number one, and let’s see what works and what doesn’t work.” And actually, that worked extremely well, because I would have never developed anything in a vacuum, or I would have gone awfully wrong without Keith, for example. But they knew more computer algebra and would immediately say, “No, no, this is not the right way of doing it. You have to do it in this other way,” and so on.

For the language itself, I am 100% responsible. I was in love with ALGOL 68 at the time. I was fascinated by what I perceived as elegant constructs of an IF closing with a FI and a DO closing with an OD and so on. This syntax eliminated the BEGINs and ENDs that I always thought were ugly in other languages. The BEGIN-END is too generic, the IF-FI, DO-OD tell you immediately what they are. We had big discussions with Niklaus Wirth in on this topic...

HAIGH: Had ALGOL 68 been in use at Waterloo?

GONNET: No, actually ALGOL 68 was the language that I used extensively to describe my algorithms, which actually never used it in the final book. I moved to Pascal in the end because of lack of good implementations of ALGOL 68. I don’t know, I think that the world has proven me wrong in being such a fan of ALGOL 68, because ALGOL 68 didn’t go anywhere in the end. It was too complicated, poorly implemented, poorly understood, design by committee—you can make all sorts of critiques. As a matter of fact, I would say that by now ALGOL 68’s only significant impact as a language is probably Maple. I may be wrong, but I think that no other language in current use has a syntax that is so close to the original ALGOL 68 syntax.

So I’m responsible for the syntax. I’m responsible for the kernel. I wrote the kernel. This idea of having a kernel in a language which is the interactive language, and the language that will become the library, I think it was my idea, but I don’t view it as a huge step. It was clear to us that we were going to have to write libraries from day one, that everything was not going to be written in the kernel, and that the kernel was sort of for the selected few that would want to code in B, or in C later. The general developers of Maple coded in a language that was a mathematical

language and not an implementation language. We were also of the opinion that the user interactive language and the mathematical development language should be the same.

HAIGH: So, reading again from the early paper, it says, “The kernel includes the interpreter for the Maple language, basic arithmetic (including polynomial arithmetic), facilities for tables and arrays, print routines (including two-dimensional display) basic simplification, and basic functions (such as *coeff, degree, map,* and *divide*.” [Bruce Char, Keith O. Geddes, W. Morven Gentleman, Gaston H. Gonnet, “The Design of Maple: A Compact, Portable, and Powerful Computer Algebra System,” in *Lecture Notes in Computer Science 162: Computer Algebra*, edited by J.A. van Hulzen, Springer-Verlag, 1983, 101-115, page 101].

GONNET: Yes, that’s pretty accurate.

HAIGH: Had that been pretty much the contents of the kernel in the very first version?

GONNET: Yes, probably.

HAIGH: So was the idea that a copy of the kernel would stay resident in memory through the whole of the user’s interaction with the Maple session?

GONNET: Correct.

HAIGH: And if several people were using Maple on the same machine, could they share a kernel, or did they need a copy each?

GONNET: No, they needed a copy each. We did not have multiple threading at the time, not even sharing of binaries at the time on that Honeywell system. So technologically it was not possible at the time. So the question of multiple threading is a much more modern question, and the kernel was designed without multiple threading in mind in that it uses lots of common memory areas and so on. But even nowadays, it’s difficult to make Maple multiple threading.

HAIGH: You’ve said that interactiveness and accessibility to users on an affordable machine was the driving motivation. In practice, was the main way you achieved that by making the kernels small and responsive?

GONNET: Right. I think that at the early stages, there was a significant amount of effort put into making the system small and as efficient as possible. It was a daily concern that we wanted the system to be as small as possible, as efficient as possible. Because we always had problems that we could not solve with the current machines—we always had problems that, “Oh, the machine is too small or it’s too slow.” So our consumers were always demanding. And this is a time that minicomputers are about to appear, actually were appearing at the time. I think that with the first grant, we used some of that money to buy a computer. That was almost a miracle, right? Suns had not appeared yet, or had appeared but were still not distributed. We had our first Sun a lot later, actually.

HAIGH: I don’t think Suns would quite have arrived then. I think the Apollo was the first really successful workstation.

GONNET: The Apollo was one. One that we bought was a machine that was made in Ontario; it was called a Spectrix. I think that the company disappeared, but we had one Spectrix. It was basically a Unix machine. That must be 1982 or 1983; not in 1980. 1980, we were just Honeywell based, and probably for the next year or so we were exclusively Honeywell based. It was only later that we ported to C, and for quite some time, as a matter of fact, we were able to use the same source to compile in B and to compile in C. It was a peculiarity of the system.

The inference from ALGOL 68 was for the Maple language itself, and I think that everybody in the group was quite conscious about designing a language that would be nice to use for computer algebra, because we were the main ones to use it. And I think that that was a very wise decision in retrospect, or a very fortunate decision, to have the language that the users are going to use to be the same language that the programmers that designed the language are going to use. That was not the case for Macsyma, nor for Reduce, nor for ALTRAN, of course. So now the people that are developing the system use the same language as the people that are going to use the system. So whenever you don't like something, you fix it—you don't suffer through it. And consequently, the users' language has all the advantages that the designers wanted to have, and vice-versa. If there is something painful in the language, the designers immediately see it, immediately become uncomfortable, and want to fix it.

HAIGH: So by that, do you mean that the Maple libraries would be written in Maple?

GONNET: Right. So Maple has always kept this architecture of having a relatively small kernel and a relatively large library. And as a matter of fact, the transition between kernel and library is very transparent. You have to really know about the system to figure out whether a function is in the kernel or is in the library, and we have migrated both ways. We have taken things that were in the kernel out into the library because we want to give them more generality, or they are not important for efficiency. Likewise we have migrated things that were in the library into the kernel, mostly because of efficiency: this function is really crucial, it needs to run very fast. In the library, there is a high penalty for executing this interpretive code. In the end there is a high penalty that's moving to the kernel. But this transition in Maple and in Maple descendents is quite transparent, and you can go one way or the other with the user not really knowing that his favorite function stopped being a library function and is now a kernel function.

HAIGH: Were there any other programming languages or systems around at this point that influenced you in this design decision?

GONNET: Waterloo had quite a bit of a tradition of designing languages and designing systems, and I would credit a lot of this to Morven Gentleman and to Mike Malcolm. I don't seem to remember any language that we copied from, but I would believe that this was sort of the folklore knowledge of this group of people all working in language design. They would say, "Of course you want to do it this way, of course this is obvious that you want to do it this way," and there is quite a lot of wisdom, in the end, in a group of people that are all working and are all very good at language design. I think that the project profited from being surrounded by people that knew about language design and would immediately say, "Oh no, this is crazy, you are doing the wrong thing."

I have to say that another important aspect I want to mention was the fact that the cycle between design and implementation was extremely short. We would test something, somebody would say, "No, no, this is bad. Change it," and we would change it right away, because we were working on the kernel every day, it was open to everybody. It's true that I was doing most of the work, but it was open to everybody. Our grad students were working on it. You want something, go and change it. It's good, fine, we adopt it; it's bad, we toss it out. We used to have meetings on Friday mornings, and we would discuss a new feature, and Friday afternoon it was implemented. And maybe Friday night it was already discarded because it was a bad idea, because we hadn't seen something. Or perhaps the contrary: it was a good idea and it was further improved the next day, or next Friday. So this very quick cycle was quite important for making the language very flexible and very usable at the end. Because if you don't like something, you

change it. And if you have to go into a design phase that you design for a month and then you implement for a year, you just won't have this flexibility. Maybe you are going to design things better, maybe you are going to avoid some pitfalls, but you are never going to define something that is pleasant to use.

HAIGH: And did that process you've described apply to the kernel itself as well as to the Maple code?

GONNET: No, the kernel has always suffered from being very much for our own consumption. For all sorts of reasons, we never released the kernel to the public. The kernel is still private.

HAIGH: And in those very early days when it was just used within Waterloo, would that have applied then as well? The users wouldn't have seen the code become…

GONNET: Most users have not seen the code for the kernel, that's correct. Our own people were obviously working in the kernel, our own group, but not even everybody. We would had our own academic group divided in such a way that say one third was working in the kernel, two thirds were working in the libraries. For example I think that Keith never touched the kernel. On the other hand, a student of mine, Mike Monagan worked so much in the kernel he definitely spent a similar number of hours to me. So some people worked a lot in the kernel, but I wouldn't say that it was more than one third of the group that was working in the kernel. And the kernel had always had this notion that it was not for everybody. It had some difficult things in it. For example, garbage collection was a very difficult and a tricky part of the code. It was done very efficiently. It has a very peculiar algorithm based on generations of data structures. If you make a mistake, the system dies in a very unfriendly way and nobody wants to fix a garbage collection bug! So the fact that we were not distributing the kernel, the fact that the kernel had to be modified so often to port to other systems, and so on, made the kernel for our internal consumption more than anything else.

HAIGH: Now while we are talking about the kernel, I was wondering if there was anything in terms of its data structures, its use of hashing, and so on that you would consider innovative. Also any aspects of the design you could perhaps link into your knowledge of algorithms, where you might have made a choice that would be distinctive or personal.

GONNET: There are several choices that you have to make when you write the kernel. We've made some choices, and those affect the system in a way that the system will never be able to modify these choices. I think we made the right choices, but this is again, like saying my religion is better than your religion. We made the decision very early of having unique representation of all expressions and sub expressions in the system. This means that if you have the expression x+1 in one place, any other x+1 is going to be pointing to the same data structure, so that x+1 or 1+x is represented only once in the system. This obviously has very desirable properties when you want to compare expressions, because if you want to compare two expressions they have to be the same point.

You also have very desirable properties when you have an expression that grows in a very repetitive way. You have some expressions of the same thing, which is typical of computer algebra when you compute derivatives. For example the division of polynomials will create the same sub-expressions again and again and again. Unique representation requires an overhead; every time that you create a data structure, you have to make sure that the structure doesn't exist already in your system, and this was done with hashing. This was quite novel at the time, I would say. People were not doing that. We were actually quite harshly criticized by a lot of people that

said, “You are crazy and you don’t know what you are doing. Your system will never work.” This makes some other things very easy. For example, the “option remember” that we discussed before. The fact that when you compute a result from an expression, you may remember this result to avoid computing it again, which in many cases is very desirable property. It is very easy to do because now your input expression has to match exactly because there is only one copy of each expression in the system. So it has to match exactly what is in the table.

I wouldn’t say that this unique representation was a novel concept. People knew about unique representation, but it was like, well, it has all its pros and all its cons. Somehow we viewed it in a different way and went for it, although most people would not go for it. And I think that I am quite happy about that decision because it meant that most of the problems were solved with much smaller data structures because you don’t have any useless duplicates. Everything that is in the system is there only once. So if you have a huge expression, it is because everything is needed; there is nothing redundant. No sub-expression is repeated. It’s something that you cannot do by brute force; it has to be done with careful data structures. So we had to devise a hashing scheme that was suitable for finding identical expressions.

By the way, the identical expressions means identical expression in appearance, not mathematically. So for example if you have a polynomial in a factor form or in expanded form, it is the same polynomial; it will be different for Maple. So x+1 and 1+x are the same for Maple, but x(x+1) or x^2+x are different for Maple. Factor form and expanded form are different. It could be done in the other way, but in that case it is clearly undesirable because now you don’t want your factor form to become expanded or vice versa automatically. Whereas you typically don’t care too much if you write 1+x and the system turns it into x+1 because there was already an x+1 in the system, although some people criticize that in Maple: If you type a+p and the system tells you p+a, why?

HAIGH: So would this choice have been influenced by the design goal of making it possible to use interactively on a smaller machine?

GONNET: On a smaller machine. The interactivity has nothing to do with it, but being efficient and running on a smaller machine is definitely something that tilted the balance in favor of unique representation. I guess that it was that we decided on unique representation that we enjoyed more benefits than we had originally thought. I don’t think that I regret that decision. I never thought it was a bad decision. I always thought that we had gotten more benefit than anything else.

Maple was also viewed as very rich in data structures internally, and also at the user level. So you can have a number, you can have a rational, you can have a floating point number, you can have a name, you can have a string, you can have an index name, you can have a polynomial, a sum, a product, a function. There are about 30 data structures that are commonly used in mathematics that are all represented in Maple. And those are represented in the kernel. At the kernel level they all have their own structure. So that was also a difference compared to LISP, for example, that had one data structure. So I think we have profited, by and large, of this ability of representing many different data structures inside the kernel in giving the user a relatively rich choice of types that can be used.

I think that there were other decisions that had to do with the language that are also very important that affected the way Maple developed that are more language type of things. For example, Maple is very object-oriented in a slightly different way than what the object-oriented

community calls object-oriented. Remember that object-orientation was not a term that was probably even coined in 1980.[1] But we had this notion that a Maple variable can hold any object that you want. The object that you put into in any variable always knows what it is, and it can be tested. So you can put an integer into x, and now x will be an integer, but you can say "is x an integer?" "Yes it is an integer," or, "no it is not an integer." Or into the same variable x you can put a floating point number or you can put a polynomial or you can put an array or you can put a list or a set—anything that you want. So the variables become like boxes that can have any object that you want in them. And you can pass this object around. This object can be stored, passed in as parameters or whatever. And whoever receives this object can decide, "Do I have this type of object? I'm going to do this." "Do I have this other type of object? I can do that." This is a very primitive form of object-orientation, but for 1980s this was sort of an advanced form of object orientation.

HAIGH: So that the type is defined dynamically by what you assign the variable to, and it can change if you reassign it?

GONNET: Right. And there is a type function, which is an active type function—it is not a static thing that you say it's the type of this and then compile this code. It is a function that is executed in run time and it typically will determine, is this object of this type, yes or no?

Another thing that was normal at the time was that we identify types as being structures themselves, so the types at first were only the arguments of the type function. Is this object of this type? But then very quickly the type became objects inside the system. Nowadays you can do with types whatever you want. You can pass them on, you can define them in a function, definitions so that only this gets accepted and so on and so forth. So types became objects inside the system as opposed to just mechanisms of the programming language that you declare, such as Boolean whatever, integer whatever. In C there can be Boolean objects, but there is no "Boolean" as a type. Boolean is a definition in the language. In Maple, Boolean is an object that I can pass around, and other functions may say is this type of whatever the type should be, and you can do this type of matching.

HAIGH: And was it possible for users to write their own types or were they limited to the ones that were…?

GONNET: Yes, actually the big thing was that the type system defined the atomic type, let's say, but all those types can be built up. So you can build sophisticated types from the components and users can build new types. Probably there we are talking about several years after the original system. The original system was obviously quite primitive. It was put together in a few weeks. Maple was very dynamic at the beginning. We never had this notion of versions until version 2.0 that we decided, okay, we are going to have versions now, but we could not really have version 1.0 so we decided it should be good to have version 2.0.

HAIGH: So you've said that the design of the package was very much influenced by your idea of what users would want, and that this was modified as they were vocal in giving you feedback. So what was the original user body that would have started working with this first version after you spent three weeks creating it?

[1] The term "object orientation" appears to have been coined by Alan Kay in conjunction with his creation of the Smalltalk language at Xerox PARC during the 1970s.

GONNET: After three weeks it was probably mostly the people inside our department that were using it. We were in the faculty of mathematics, so pretty soon Maple found it's way into several courses. We had a computer algebra course, but that was an advanced course for people that wanted to know what computer algebra was. But Maple started finding its way into undergrad math, into some other courses, at some point into economics courses and so on, where people were consumers of mathematics. For some time there was a contest to see how Maple will do in a final exam of algebra or of calculus. And it was quite interesting because Maple was getting better marks every year in those exams, but also the exams were changing because teachers recognized at some point that there was no point in asking questions that a computer algebra system could solve. Maple was helping in the teaching, but it was also showing the teachers that there was a powerful tool here, and the classical type of questions that were asked might not make sense any longer. You don't want to ask, "what is the integral of this function." That's a typical thing that a computer or algebra system will answer for you much better than you could do it by hand. You need to understand how to integrate and you need to understand the fundamentals, but do you need to be a robot that integrates anything? The answer is probably no. It's the same way that you need to understand how to add two numbers, but it's pointless to give an exercise in a final exam, "Add these two numbers," that are very long and have lots of carries. No. That's why we have calculators these days that do the job very well.

I am probably collapsing many years of history now, but that was something that happened almost from the beginning: that Maple got it's way into the teaching of mathematics, not only the teaching of computer science. It got us input: what we needed to do, how we needed to change the system, this is not reasonable, why do this, and this is wrong, or this is wrong in certain aspects of mathematics. But it also affected everybody by showing what it was possible to do automatically, showed what things were foolish to ask the students. And I think that now everybody recognizes that, but for quite some time there was sort of a self-adaptation of both sides: the teachers on one side and the development of computer algebra.

Maple, in some sense, was the first computer algebra system of which we could say with a straight face that was useable by a large number of students. It was just not possible to give MACSYMA to 100 students, to try your exercises using MACSYMA or using Reduce. It was just not possible, and with Maple it was possible. At first it was a little bit painful, but with computers getting faster and cheaper it became possible.

HAIGH: When you talked about getting feedback from the early users, would that be feedback directly from the students or from the instructors?

GONNET: Students, instructors, and colleagues who were using Maple for doing numerical analysis or doing physics or doing something or other. We had a department of applied mathematics in the math faculty, they were also big users of Maple.

HAIGH: Would faculty members at this point have been using it as a tool in their own research?

GONNET: Yes. I wouldn't say extensively, but yes, many people were getting into using computer algebra.

I want to mention something here which is very peculiar about mathematics, and I think that computer algebra and Maple in particular and Mathematica to some extent have contributed. There is a huge body of mathematical knowledge. That is indisputable. Advances in mathematics occur, but the weak link of mathematics is in trying to make this knowledge available to everybody.

If you want to solve a differential equation or something like that, you can go into the literature and study and eventually you may or may not find something that solves your problem. But you have to do a very long and extensive search, and it usually is quite difficult for anybody who is a consumer of mathematics. We had a situation where it is very hard for the consumer of mathematics and the mathematical knowledge to meet each other. Sometimes actually the answer is so complicated or the answer doesn't exist, and so it's just not possible to connect the two.

You're not going to solve all the differential equations of certain problems. If you are studying differential equations, you are going to study some pattern of differential equations. You don't solve every problem. So now a user has this particular differential equation, the best that the user can do or the consumer can do is read all these papers and extract the methods. You see computer algebra somehow made this bridge possible. Computer algebra is a tool that allows a user to access the big body of mathematical knowledge, and I think that in some sense, all computer algebra systems owe their success and their acceptance to this thing that they do. That is, they provide the bridge between mathematicians doing mathematics and users needing the mathematics. Nowadays, if you want to solve a problem you go to Maple or to Mathematica, you describe your problem and most likely somebody has embodied the mathematics into an algorithm that solves your problem. Before this, your only choice was going to go to the library and do an extensive search, and eventually have to decode these mathematical papers and translate them into your own language.

HAIGH: Actually that does bring to mind a question about the different experience a user would have using a symbolic package like Maple and a numerical one like Matlab. If this user that you are talking about who has a problem that they need to solve, would you say that they would learn something different from attempting to solve it symbolically than from attempting to solve it numerically?

GONNET: I have always been surprised at how little users understand the difference between purely symbolic and numerical, and actually how easily they confuse each other: how many Maple users use Maple only for its numerical properties and how many people use Matlab very cleverly to extract symbolic answers. (well, Matlab now has parts of Maple in it). But even by approximating or by plotting or by using some of the primitive facilities that they have doing some symbolic computation, a lot of users just see the systems as doing mathematics for them, and for them whether it's symbolic or numerical is a blurred line. They may not even care!

HAIGH: As I understand it, if you tried to use a purely symbolic package and it couldn't solve your problem, it would give you some kind of error; and if you use a numerical one it would give you an answer but it might not be meaningful. Would that describe the situation faced a naïve user who didn't know anything about the mathematics and was faced with the two?

GONNET: I think if you have a naïve user that doesn't know any of the mathematics, they're probably cooked in both systems. We have to recognize these systems require some knowledge of what you are doing, no matter how much we claim that they do mathematics for you. If you have problems understanding what is the difference between an integral and a derivative, Maple is not going to help you, Mathematica is not going to help you either, Matlab is not going to help you either. You have to have some basic understanding of mathematics to use these systems.

We used to say that Maple and computer algebra was like a very diligent slave. The analogy is quite good, actually. It's a very careful, very stupid, but very diligent slave. You say add these

two polynomials and it will add them without making a mistake. Integrate this, and it will integrate applying the rules very carefully without making mistakes. But the driver is the person that is requesting the things. It's true that the systems get better and better and better. The more packages that we have and the more that the systems get into special areas of mathematics or science, the more you can pose the problems in the terms closer to the original area. Then the system is able to translate them into mathematics and give you an answer that makes sense. But I would insist, again, that these systems, in the end, have to be used by someone who understands the mathematics. For someone who doesn't understand the mathematics, is just going to be a GIGO system—garbage in, garbage out.

[Tape 2 of 6, Side B]

HAIGH: Now I just have a few more questions about this very early version of Maple before we move on to talk about later developments. I believe you described the contributions of Morven Gentleman and Keith Geddes. What was Bruce Char's contribution?

GONNET: Bruce Char was not there yet. Bruce Char was hired some time after the project was started, and he immediately became a main contributor to the group. It was very nice to have Bruce Char because Bruce had some very serious experience with MACSYMA. "No, no, no, you're doing it wrong. MACSYMA does it a lot better." Which was very annoying sometimes, but it was very good at keeping us honest in what we were doing. He contributed very much to the design, in the algorithms, and he had lots of students that were working very hard in the project. He was a young faculty member. He came right out of his Ph.D. to join Waterloo, and almost instantly became a strong contributor. But he was not there for the initial design

HAIGH: After he arrived, did he become the main contributor on the kernel or did you continue to handle that yourself?

GONNET: No, actually, the kernel was really my domain most of the time and when Mike Monagan became very active in Maple, he was probably working as much as I was working on the kernel. And other people also worked on the kernel. But in terms of faculty members, it was only me. It terms of grad students, it was mostly Mike Monagan. Stefan Voerkoetter was another master's student of mine that worked on the kernel quite a bit.

HAIGH: Now, you'd said that quite quickly after this initial release, you were able to get a small grant to fund continued development. So what did that grant pay for?

GONNET: If I remember correctly, that grant paid for some equipment and for two people, two students or two staff that we hired, basically to help develop the system. And it was at this time that we sort of officially declared that we had this Symbolic Computation Group, the SCG, which was sort of the umbrella under which everything else was happening.

HAIGH: So that group didn't exist prior to the Maple project?

GONNET: No.

HAIGH: As time went by, did it gain any other significant activities?

GONNET: No, it has always been mostly around Maple. A model that I have used and Keith Geddes has used quite extensively, and that I like and defend very much, is the notion that a Ph.D. student should develop some science, some new knowledge, and when it's in computer algebra, this knowledge has to be encapsulated in some facility in the system. So you work out a

better method for factoring polynomials, by golly you have to have an implementation that does it. I don't want to know about nice theorems that are not implementable.

It's my peculiar view. But this has been quite generalized, and I think it also gives quite a bit of satisfaction to the student, because they see that what they do gets embodied and comes to life. Anybody now that wants to solve this particular type of differential equation are going to be able solve this differential equation because the work that I did. And so it's like making your mathematics alive. Which goes back to what I said before, that computer algebra has brought this necessary bridge between consumers and producers of mathematics. I want to give my students a practical orientation, that if you are doing good science, the good science is also to be matched by some real code that will eventually serve make what you have discovered alive for other people.

HAIGH: With the two students you were able to hire, was that when it got converted from B into C?

GONNET: No, the conversion to C was done by another student, Howard Johnson, who later became a faculty member and worked with us. Actually, Howard Johnson made several contributions to Maple. Because Howard was doing this conversion, he decided to work on a macro processor that was powerful enough to convert to one or the other language. So we had some source code that was slightly above the B or C, and it was preprocessed by a preprocessor (very similar to what we have nowadays with a standard C preprocessor) that had some extra features. This allowed him to maintain code that was simultaneously executable in B and in C. With time, only the C version was maintained, and eventually all that was discarded and only the C version became the only one.

There's some interesting naming in there. The name of preprocessor language was MARGAY. We used to say the kernel is coded in MARGAY. MARGAY is this language that with macro processing will go to B or will go to C, depending on the macros. (margay is a type of wildcat that is not very well known. We once made a little excursion to see a margay. I think it was in Arizona in the Desert Museum or some place like that).

You also have to understand that in the early 1980s there was no standard for C, and the C compilers had very different levels of how many features they accepted from the language. In particular, many early compilers did not accept structures. B didn't have structures; early compilers of C did not have structures. Consequently, the kernel of Maple did not have any structures. So from that point of view, the C code is a little bit deficient, technically speaking. It's all coded without structures. It's all coded in a hard way, in some sense. If we were coding it nowadays, we would definitely use richer structures. But this was something that was forced on us by the languages that were available at the time. C was not even available when we started the project.

HAIGH: It says in the paper that it was ported to C in 1981 when Waterloo acquired a VAX, the original version having been for the Honeywell. [Bruce Char, Keith O. Geddes, W. Morven Gentleman, Gaston H. Gonnet, "The Design of Maple: A Compact, Portable, and Powerful Computer Algebra System," in *Lecture Notes in Computer Science 162: Computer Algebra*, edited by J.A. van Hulzen, Springer-Verlag, 1983, 101-115].

GONNET: Okay, '81, that sounds very early. I would have said that it was '82, '83, but if it says in the paper '81, that would be great.

HAIGH: That would be prior to receiving this initial grant.

GONNET: Yes.

HAIGH: So when you got the grant, it was already in C.

GONNET: No, sorry, sorry, it would be concurrent with the grant.

HAIGH: What other things changed between this very early version produced in three weeks and versions that would be in use within Waterloo, say, two years after that?

GONNET: There were thousands of changes, probably. But it was sort of an every day type of evolution. It was not something that, oh, we completed version one now; we're going to version two. It was sort of a natural evolution. I think I can probably pinpoint a couple of events. At some point, we redid the entire internal arithmetic. We had made a bad choice by thinking that people would never work with more than 500 digits; I think it was some number like that. Compared to all the systems that we had, it was unthinkable that anybody would want more than 500 digits. And it was clearly wrong, some of those decisions that you make, and we needed variable precision arithmetic. Because of the technicality, it was an intricate change. I think that most people would not even remember that we didn't have variable precision arithmetic in the first place.

We didn't have garbage collection for quite some time, and Maple was quite economic in memory and very CPU-intensive. Even though it appears to be crazy to say we didn't have a garbage collection, this was not a serious flaw at the end. We were running a significant computation without garbage collection. Maple was very capable of dumping all its memory into a file and then restarting, so that was sort of the "garbage collection" approach. If you were really computing something very large, you would sort of dump everything in a file and restart. It was only in 1984 that we had bona fide garbage collection in a production way. So you see it's three years that the system went without garbage collection. We also had very primitive ways of printing at first, which was basically all line-oriented. It was a big thing when we started printing in two dimensions. Again, it was very primitive at first, but then was quickly recognized that two-dimensional printing, even in ASCII characters, was very important. We're not talking about big map displays; we're talking just about plain characters displayed in a fancy way on the screen to mimic an integral or a summation or whatever.

HAIGH: Actually, that brings up another question. Given that everything is occurring in ASCII, was there a standardized way of representing complex mathematical notation in ASCII, or did you have to come up with your own notation?

GONNET: No, we had our notation, and that was a natural extension of the language. The language allows you to write expressions, and so the way that you write expressions becomes a way of describing mathematics.

But maybe at some point we should talk about OpenMath, which was an effort that I started. I started in earnest when I came here to Zurich, but we were talking about ideas that were leading to OpenMath for a very long time. And yet, they never crystallized very well. The notion that we have to have a language that allows us to transport mathematics from one place to another is a very important notion, and it wasn't realized for years, and only recently with OpenMath becoming more significant, it has a chance of becoming a reality. But it has always been the case that you have either Maple or you have Mathematica or you have whatever else, and it's very difficult to transport your mathematics from one to place to another, or your lab expressions or whatever.

HAIGH: Okay, I'll make sure that we cover that in a later session, then. So from your point of view, the three main areas of expansion of the kernel from this initial 1980 release through about 1984, would have been: the addition of garbage collection, the improvement of the arithmetic in terms of variable precision and an increase in the maximum possible number of digits, and the ability to print in two dimensions.

GONNET: Yes. All those are still eclipsed by the growth of the library and the growth of the general system, except that those things are incremental, right? So those are points that we can say that was a significant step. But all the steps are still very small compared to how the language grew.

HAIGH: Let's talk about the growth of the library, then. Now I think you'd implied earlier that a lot of the growth, in terms of functions would be coming from the uses themselves.

GONNET: Yes, but not exclusively. A lot of the growth was coming from us.

HAIGH: And by "us" you mean members of the Symbolic Computation Group?

GONNET: Members of the Symbolic Computation Group or their employees, or their students. So a significant growth came from Ph.D. students that were working in the group, that were contributing their code.

HAIGH: So as the system developed, how many Ph.D. students might have been involved with the group?

GONNET: About 20. Now, I am probably making an injustice there because in total, there must have been more than 20. But at some point, the company, Waterloo Maple software, took control of the entire development, and the production of Maple code became less dependent on the research groups. If I had a student doing computer algebra, it's not necessarily a requirement that he write something for Maple. And it's not guaranteed either that the company's would include this in the package.

HAIGH: So in this period through 1984 that you had mentioned earlier, prior to commercial distribution, what would your sense be of the rough balance between contributions from members of the group and those from people outside it in terms of size of code or number of functions?

GONNET: I would say three lines produced internally by professors, students, or staff for one line from the outside—three to one. It's more than two to one. We were very active during that time. It was Bruce Char, Keith Geddes, our students, later George Labahn joined. Howard Johnson was working part-time. Benton Leong was another faculty who was working part-time in computer algebra. It was a lot of people that were working in the project. And it was also very a high-profile project at the University of Waterloo, so students were very interested in working with us. It was very attractive for the students to work in Maple. A lot of students had this idea of semi-magic. You'd type an integral, which is a difficult thing, and out comes the answer. This is magic! Somehow, a lot of mathematics, in particular, integration, has been taught to a lot of people like a bag of tricks. A mathematics professor will know how to integrate all these complicated functions. In modern times, it may no longer be true. There is still that aura that the selected few can integrate functions. And to have a system that does it automatically is a little bit of magic, right?

HAIGH: So would many of the contributors have been students who were taking mathematics courses and had been impressed by the system and were interested in working more on it?

GONNET: Yes. I think still most of our students were from computer science, although we had some students from applied math and some general students from mathematics. But most of our students would be from computer science.

HAIGH: At what point was the package first distributed outside Waterloo?

GONNET: I think it was distributed very early outside Waterloo—very, very early. The first commercial distribution was in 1984 where we arranged with a company named Watcom to do the distribution.

HAIGH: Prior to that, had there been non-commercial distribution?

GONNET: We were charging some money because needed to pay for the tapes and to pay for somebody producing the tapes and somebody sending the packages by the post and so on. So we were charging, say, $200 or something like that to do the service. So we were charging but could not be considered commercial distribution. I have to say, we were instantly successful with that. Cheques were arriving. This was inside the university, and it was very difficult for us to justify receiving money and it was difficult for justify to put some staff to do things that were clearly not academic. Making tapes and sending envelopes was clearly not academic, but you couldn't give it to a secretary either because it requires some knowledge of the system—you have to compile, you have to get it ready for particular installation or whatever, get all the pieces together, make sure that it installs and so on. So this is a dilemma that you always have in a university when you develop software. There is a stage, if the software is successful, that you have to start distributing it. No structure in the university normally will allow you to pay for the down-to-earth costs involved in distributing. And nobody likes that you're receiving cheques from somebody else because the university administration is worried that they are getting into some commitment that they will not be able to honor. They are also worried that you, all of a sudden, have money that they cannot account for and so on and so forth. So universities are very ill prepared for this type of thing. It always works for a short period of time, if it works at all, and then it has to find some other avenue.

HAIGH: Prior to the point in 1984 where Watcom began to distribute it, do you have a sense of how many other sites might have been using the software?

GONNET: It's difficult to say because we had distributed hundreds of copies, but how many of them were being used is difficult to say. You always like to inflate the numbers a little bit and say we had so many users. I would say that we had in the hundreds of users. I wouldn't say that they were in the thousands. I am sure that some people received the software, tried to install it, didn't work for whatever the reason, never used it again. I am also sure that a lot of people were making very good use because we were getting the feedback.

HAIGH: So a lot of this feedback and contribution of library functions already would have been coming from the users outside Waterloo.

GONNET: Right. Another measure of people using the system is from the books and material that you produce, and so we had some technical reports that were describing the language, and we would print 100 copies and we would run out; print another100 copies and run out. So obviously either some people were wallpapering in their basements with the technical reports or they were using them.

HAIGH: It's your sense that these external users would have been primarily using it for teaching purposes rather than research?

GONNET: Both. I think that some people were doing their research. Some people were probably curious how much you can do. I guess a lot of people were frustrated also because they would throw it a difficult problem, the system will not do it, and they say, “This is a toy” Understand it or not understand it, the computer algebra is a very difficult problem to solve in all its generality. And also understanding that there were no standards, right? It was difficult to say, “The system has to do this.” If you have a spreadsheet, you expect it to compute, period, right? It’s deterministic. It computes predictably. You expect an answer and you can determine it. If it has a bug, it doesn’t do what it’s supposed to do. In computer algebra it is not the same. If you give this differential equation, the system says, “I cannot solve it.” Now you don’t know if it doesn’t have a solution or the system is not smart enough to find it, or that by massaging it sufficiently you will be able to convince the system to find a solution. So this is what I mean by the lack of a standard. You cannot say, “The system is perfect.” There’s no other system that you can say, “System X solves all these problems you are not solving.” There was no System X. And in general, we were better than MACSYMA very soon, I should say. In some sense, MACSYMA for us was never the competition.

HAIGH: So by “better” you mean that, although the goals were originally compactness and efficiency and usability had been driving it, that Maple had quickly come to solve a wider range of problems than MACSYMA?

GONNET: Yes. Almost instantly, we were able to solve bigger problems than MACSYMA was able to solve because of our compactness. MACSYMA had quite good libraries because it had many years of people working, but it was also a nest of bugs because different people had worked at different times and coded something and left. MACSYMA was extremely difficult to maintain, and so getting the wrong result in MACSYMA was not extremely surprising. MACSYMA had also this extremely unpleasant behavior that, in the middle of a very long computation, it would spit out a huge expression and ask the user: “Is this expression greater than zero or less than zero?” No idea whatsoever. You know, you get half a page printed, and it asks you a question. Well, how do I know? And of course, the system was deciding that it needed to know whether this was a zero or not, or a positive or negative, and it was asking a question. Those types of things were design mistakes that MACSYMA could really never shake off. It may have been a mistake on our part, but we never considered MACSYMA to compete with us. We assumed automatically that we were better. At first, maybe incorrectly, maybe we were not better. But in the end, we were definitely better than MACSYMA.

HAIGH: This is jumping ahead slightly, but my impression was that MACSYMA, as it was sold in the early and mid-‘80s was tied by Symbolics to its proprietary range of hardware.

GONNET: Correct, because MACSYMA was LISP based, right?

HAIGH: Yes.

GONNET: And Symbolics was selling these machines that were LISP machines. A Symbolics machine was basically a LISP machine. And actually one of their strong selling points was that they would sell MACSYMA when there was no competition, of course. But they were also selling a very nice environment for developing LISP programs, but their goal was to sell the LISP machines.

Symbolics was located near Kendall Square in the heart of MIT, so they had very close ties to the Artificial Intelligence lab. It’s no secret. And so for quite some time, they either had the original people from MIT that had developed MACSYMA, or had easy access to all of them.

The company at some point got into trouble. It reached a point that they had so much code that they could just not maintain it. This is always a danger for any software, to become critical in the sense of you have so much legacy code that you don't have enough people to maintain it. And I think that that's what happened to them more quickly than what they expected.

Now, they were a real company. We were not a real company; we were an academic project. So they had to advertise and they had to sell for high prices and so on. We were not advertising. We were just distributing the software for peanuts compared to what they were charging. So we never considered that we were competing with MACSYMA. That's the truth. Whether they were thinking that we were competing with them, I really don't know. Maybe they were not even thinking that we were competing with them. But in the end, MACSYMA disappeared. Maple didn't do very well with Mathematica either, in the end of the story, but that was our own mistake.

HAIGH: Did the separation you talked about between the kernel and the libraries help Maple deal with this problem of maintaining a large volume of code?

GONNET: Yes, absolutely. It also made the code transparent, so we had lots of people that were users of Maple that would say, "This function is very nice, but you have a bug here. Fix it," and they would even send us the fix. That was because it was in the libraries, because these were people that needed the math done, and if it was not doing it, they would go in and fix/improve it. They had all the tools to solve it—they knew the language and they had the code, so they could go and fix it. If it was in the kernel, they could not fix it. But if it was outside; and most of the code was outside, they could go and fix it.

HAIGH: So I'm seeing a parallel between that and the ethos of today's open source software movement.

GONNET: Yes. Actually, that's quite correct. I was not the only one, but I was definitely adamant that the library should be accessible by everybody from the beginning. At some point, we were not distributing the library, but I had made sure that you could print any function from Maple. Well, you would get it without the comments, but at least you'd get the function. So the library was never a secret. The library was there. It was copyrighted, yes; it was not free. But anybody that wanted to know how to do things or how to fix things could be able to do it. And as a matter of fact, it did work very well in the same way that open software works nowadays. That has two aspects, which people usually don't recognize right away. One is that you know what you can contribute; you see the software, you fix it or you add software to it. But the second aspect is that it has a component of teaching. By looking at how things are done, you learn something. You learn the style, you learn how to do it. And in the case of Maple, that second component was very important because people didn't know how to do it. How do you find an integral? Well, there are methods. There are algorithms. How do you factor a polynomial? And so people could go and see how things were actually done and mimic it, in some sense. "They are doing this in this way and I want to do that, but it's similar. I can now extract from that code and produce my own code." And that's very helpful. So having the library available I think was quite a positive thing for Maple.

Maybe we should have had the kernel also, and that would have probably kept us more honest about the quality of the code of the kernel. But the kernel was more delicate, and because it was delicate, we always thought that if we were releasing it, people would just break it left and right and would not invest the amount of time to learn all the details of the kernel. I'm still thinking

that that was the right decision. The kernel was very small all the time. It was doing the simplest things, or the things that needed lots of speed. Nobody's challenged by the fact that the kernel does addition, subtraction, division of integers and floating point numbers and expansion of polynomials and so on. You are interested in knowing how an integral is done or how to solve an equation.

HAIGH: I think you implied a moment ago that in the early days, even when you were duplicating the tapes yourselves and sending them out, that there was a copyright attached to the code.

GONNET: Yes, there has always been a copyright attached to the code. The copyright has always been there. The library was distributed, but it was copyrighted. There is no contradiction on distributing for free, but still keeping the copyright.

HAIGH: Certainly. Was that a university policy or was that a decision that was made by the group itself?

GONNET: It was a decision made by the group.

HAIGH: Was that because you thought the code might have some commercial value or was it to stop other people from making money out of it?

GONNET: A mixture of everything. You also want some recognition, too. Some of the functions were really hard work by members of the group, so you wanted to say, "This has been written by such and such," and want the recognition.

HAIGH: Can you remember if this self-distributed version was accompanied by any kind of license agreement?

GONNET: Oh yes, there was definitely a license agreement where, among other things, the university wanted to put some language that they would not be liable. Pretty quickly people recognized that you could do a lot of damage by giving the wrong mathematics to someone who is doing some engineering. So they wanted us to protect ourselves. So there was some standard language that no matter what Maple does, we are not responsible.

HAIGH: Would the license have stopped somebody making a copy of it and giving it to another site?

GONNET: The price was so reasonable at the time. Now you may think that $200 is too expensive because you can buy software for ten bucks. But at the time, I think people viewed it as very reasonable. Granted, we were selling mostly to academics in universities, so people were viewing the price as a very reasonable price. The number of illegal copies that you have, at a certain level, has to do with the price, of how people perceive the price. For example, we invented one thing, which was the square root law for licenses. One license is, say, $200, right? Four licenses, we are going by the square root law. So square root of four is two, so for four licenses, you pay twice the price. For 100 licenses, you'd pay the price of ten, which is the square root of 100. So it increases, but it increases much more slowly than linearly. The effect that we had with this was quite remarkable. Most people would say, "Oh, what a reasonable policy. Sure we are going to have nine licenses; we pay three licenses." As opposed to what would have happened and said, "Oh, nine times? We're going to want one license."

Anything illegal that you do with the software, I think at a certain level, is tightly related to how fair the price is perceived. And if the price is perceived to be very unfair, then people will try to

cheat, no matter what. On the other hand, if the price is perceived to be extremely fair, “These are good guys, you know? Let’s pay them what they say and be in the good books.” When we were distributing from the university (and this was happening before PCs) I think that that was definitely was the spirit.

HAIGH: Prior to the distribution agreement with Watcom, had you made any efforts to make Maple portable to platforms beyond the VAX?

GONNET: Well, we were running in most of the machines that we had in Waterloo. So we had an IBM version that was run under CMS or something like that, whatever the big IBM were at the time.

HAIGH: So was there already a C compiler available for the large IBM machines?

GONNET: Actually Watcom was developing a C compiler for the IBM at the same time, and we piggybacked on their development. So we had a version that was running on IBM machines. I remember other people ported Maple to very strange machines like large CDCs, scientific computers that were very powerful but very odd. For many, many years, VAXes were the main computing power in our university and most universities. We had VAXes running Unix, as a matter of fact; it was sort of the standard scenario for us. They were being slowly replaced by Suns, or competition of Suns, also running Unix. As I said, we had a host of small machines. The Apollos, I remember. I remember the Spectrix; I don’t remember the others. We can probably dig up some of the names. I could say that compared to other systems, we were widely portable. We would not be widely portable to today’s standards, but compared to what was happening at the time, we were very portable.

HAIGH: When I’ve talked to people who were working on numerical software libraries in Fortran, then it’s clear that in that case there were two parts to the portability: optimizing for the machine’s arithmetic, and dealing with deficiencies in its compiler. In this case, were the portability issues purely coming from the compiler, or were there any differences between machines in terms of their arithmetic or architecture that required changes in the code?

GONNET: Several answers to that: Our architecture was a blessing for porting because if we could compile the kernel and we could test it, all of Maple would run. We could not have a situation where, “Oh, you add this new library and your compiler is going to be a new version and it’s not going to work,” or whatever. We had also developed quite a bit of an infrastructure where we had a lot of our own numerical functions. Not for sine and cosine, although we did have versions of those, but for more strange functions—gamma function, error function, functions that are off the beaten path. But sometimes you would find that in porting to some machine, there was a terrible implementation of those functions. You just could not live with it and have to implement our own.

For quite some time, we were not relying on any floating point hardware for anything other than floating point computation of the system. So we were quite insulated from problems from the libraries. B and C are, in the end, are very simple compilers for which optimization is rather straightforward. So the code was usually very efficient. Yet, sometimes we tuned for a particular implementation, but most of the time it meant that once we had the kernel running, all of Maple was running. So I guess that simplified porting.

HAIGH: So it was relatively straightforward.

GONNET: I should mention that from the beginning, we had a software practice of having a very extensive test suite. The test suite was a very active test suite. Every time that we found a bug that bug, when it was fixed, would become part of the test suite. It was the regression test suite, also. So we would have the testing that programmers or coders decided, plus all the bugs that had been detected that had been corrected also. Turned out that the bugs are excellent witnesses of future problems, and so our test suite was really valuable and also really valuable at maintaining consistency with previous versions, consistency with the users, and finding porting problems.

[Tape 3 of 6, Side A]

Session 2 begins, March 17, 2005. Conducted in Professor Gonnet's office at ETH in Zurich, Switzerland.

HAIGH: In yesterday's session we reviewed both your career in general and the development of Maple through about 1984. I wonder if you could say something about how your involvement with the mathematical software community might have developed over the course of the early 1980s. You had said that when you first began to work on Maple that you had been writing the package that as a consumer you would have liked to have had, but presumably as you became more involved with mathematical software you came into contact with the community of people working in the area.

GONNET: Right. I also said, and this is very true, my main area in the very early '80s was still algorithms, I had some contacts with the scientific computing community, very good contacts. My supervisor in particular, Alan George, was a very central figure in that community. Actually there is a coincidence that my first two papers that I ever published were in numerical analysis, but that was almost an oddity in some sense. I was not really a mainline scientific computation person at the time. I was doing analysis of algorithms, and yes, I was knowledgeable about scientific computation, but it was not my main activity. For quite some time I think I did not make any efforts to become part of the mathematical community. Eventually papers started appearing on algorithms for symbolic computation with my name on them, and I was recognized as a person in symbolic computation and hence in the mathematical community. It was more of a consequence of working for so many years in developing Maple and so on that eventually you find algorithms or build something or discover something that you end up publishing in that community. You also start going to the conferences because your students go to the conference because all the colleagues are going to the conference so you end up going to the conferences and sort of become part of the community. So it wasn't so much an active effort to become part of that community as slowly being incorporated in that community. For example, I didn't go the conference that the main Maple paper was presented. I didn't go to computer algebra conferences for quite some time.

HAIGH: Chronologically speaking, at about what point might you have begun to go to those conferences?

GONNET: I guess 1984-85, or maybe even later.

HAIGH: At this point, was the computer algebra community largely separate from the numerical analysis and numerical computation community?

GONNET: Yes and no. In Waterloo there was a strong connection. As I mentioned before, the main people involved in starting the project, Keith Geddes, Morven Gentleman, and Mike Malcolm were mainline scientific computation community. Many other people in the community

had their origins in the scientific computation. However, there were people that were coming also from artificial intelligence and hence were not scientific computation as we defined them now. I don't know, maybe Waterloo was a bit of an exception in having so much influence from the scientific computation community in computer algebra. I don't think it was the unique exception, but it's difficult for me to judge. I don't remember the origins of all the people that I have seen. I would say that there were two main origins for the people in the computer algebra community, either artificial intelligence or scientific computation. Actually maybe we could also say physics—a lot of people doing physics were turning into computer algebraists.

HAIGH: You mentioned to me outside the interview that you were well acquainted with people like Cleve Moler. Was that as a result of work on Maple?

GONNET: Actually I don't know where we met for the first time. Cleve used to come to Waterloo too, so I think that we met in Waterloo. Most likely we met in Waterloo. Cleve was, I think, a very good friend of Mike Malcolm, so probably first time that we met was in Waterloo.

I think that we respected each other. As I said before, I have a tremendous amount of respect for Cleve Moler because we were both down-to-earth writing systems and we were very proud of what we were building. So neither of us had any hesitation. If there was a bug to be fixed, we were the ones that would go and fix it. We were not managers and said, "Oh, there is a bug. Go and fix it." We would just go and do it ourselves. I think that from that point of view we had lots of opportunities to discuss things that were interesting for us and common experiences and so on.

Later, Mathworks and Waterloo Maple had a commercial relationship. Actually that was architected by both of us sitting at some point and saying we had to do something together. Then it took some formal shape and so on and the contract was written and so on. But the essence of the arrangement was something that Cleve and I decided and agreed upon. We had always a very good and friendly relation. I think we never published a paper together though.

HAIGH: Just before we return to Maple. In the broader context of your academic career in general, I was wondering during the 1980s were there any associations or interest groups within which you were particularly active?

GONNET: Well, in the computer algebra community I was definitely active. I don't remember in which year, I was chairman of the computer algebra conference. The computer algebra community has one main conference that used to alternate between Europe and North America, and more recently was alternating between Europe, North America, and the rest of the world. This is the main focus for all the people that are doing research in computer algebra. I was participating in that community in the later '80s, say second half of the '80s to the extent that I was chairman of that conference. I think it was '89 that I was chairman. I think that maybe '9X I was chairman again.

HAIGH: What would that conference have been called?

GONNET: It had several names. Oh, it's embarrassing. It was EUROSAM at one point, and then it was called ISSAC for many years. I think it's still called ISSAC.

HAIGH: Were you involved with the ACM's Special Interest Group on Symbolic and Algebraic Manipulation (SIGSAM)?

GONNET: SIGSAM. Not very much. I think that we may have published a couple notes in there, but nothing more significant than that.

HAIGH: Do you know if that was an active group in general?

GONNET: No, not really. The bulletin had quite often interesting material, but SIGSAM, by being non-refereed was not a place to present your best material. In particular for the people that need to have solid merits for the furthering of their career. So the papers published in SIGSAM were more reports, conferences, "I have done this and it's interesting," but not the standard paper that you would publish in a refereed journal.

HAIGH: During this period, were there any groups outside the area of computer algebra that you were active within?

GONNET: Definitely. In 1984 Waterloo made an arrangement with Oxford University Press to make a consortium to computerize the second edition of the Oxford-English Dictionary. Starting in 1984 with a different set of colleagues, in particular Frank Tompa from computer science, we started what was called the Center for the New OED, which was a multidisciplinary center that involved people from the English department, Computer science, and some people from History. The main activity of the center was to help Oxford University Press in various aspects of the computerization of the OED. This project was tremendously successful and tremendously rewarding to all the people that participated in the project.

The goal of the project was to produce a computerized version and second edition and print it, and that was mostly the responsibility of Oxford University Press. Waterloo was involved in various aspects. We had the idea of parsing the text as a programming language. We produced text-searching software that was very fast for searching any word in the entire dictionary. We did several other tasks related to the organization of the projects and how to coordinate the various flows of information and so on. We exchanged people with Oxford University Press. We stayed in Oxford several times. People from Oxford came and taught courses in Waterloo. We organized a yearly conference that ran for about ten years which gathered a lot of people from various disciplines that were interested in dictionaries, whether it was dictionary making or lexicographers or linguists or computational linguists or computer scientists. There is no question that that project took a significant part of my attention for the rest of the '80s and in the early '90s too, although I would say that the main activity was between 1984 and 1989 probably. That took a lot of hours away from computer algebra for me. I didn't stop or anything working in Maple; I continued working in Maple throughout the time. But definitely that became my second focus of attention at the time.

HAIGH: Was your group producing software that would then be used by the editorial team in Oxford?

GONNET: That's correct. As I said, the main contributions I would say was the parsing, which is transforming the input text that was typed in into text that had structure in the structure of the dictionary, all the proper tagging. That was done by and large automatically. That was one contribution. The second part was fast text searching that was used for all sorts of things. In particular was used to clean up and to improve the dictionary, but then it also became an independent tool for searching any text database. It was actually the main piece of software that started the company Open Text

HAIGH: We'll talk about the company later. Had the impetus behind the creation of the center come from any kind of personal interest on your part?

GONNET: The Center for the New OED? Yes and no. The people from the Faculty of Arts were very excited about the project. This was a project that was negotiated by basically the authorities

of our university without very much of our involvement initially. I remember very clearly discussing with Frank Tompa. It seemed that a disaster scenario was brewing because all the people from the English Department and from History were extremely excited about this project. They all wanted to go forward. They thought it was just great that we would be in a partnership with Oxford University Press. I was seeing real computational problems that were not easy to solve. The Oxford University Press had real problems: how to input the data, how to parse it, how to produce a new dictionary, and so on. I saw nobody that would do the computational part. I was seeing sort of a collision course. And if there was a failure, the failure was going to be because Waterloo's computer science people did not do the job. I was seeing that we were going to get all the blame at the end. Computer science was going to get the blame. At the time, Frank Tompa and myself together with Paul Larson were the main people doing databases. I saw also some harm. Of the computer science people that failed, who are the ones that failed? Well, it was a database project, and the database people did not do it. I was actually worried about that.

At one point, I discussed with Frank this problem. I said, "Well, we have to do something to at least clarify our position." I said, "Well, we either do it or don't do it, but let us take the initiative and force the situation." We came up with a proposal, and we said if we get this and this and this and this, such as space for the Center for the New OED and funding for the Center for the New OED, then we will do it. If we don't get this, then we are not involved and we should not do it. So that at least it's clear. Yes, Waterloo would have failed, but computer science had said, "We will do it if we get this." They didn't get it, so computer science cannot be blamed. To our surprise, the university gave us all that we wanted.

HAIGH: So the university as a whole had won this contract?

GONNET: Yes. It was a partnership I think was the right term. It was a partnership between Oxford University Press. Oxford University Press tendered this project between 10 and 20 companies and institutions. To the surprise of everybody, the Canadians, the people in the colonies (all other tenders were from the UK), won the partnership. I think that it was a very successful project for everybody. I think that the Oxford University Press people were very happy also.

HAIGH: Did the project ultimately receive any financial support from Canadian sources?

GONNET: Yes, the University of Waterloo supported it, conditional to getting a major grant from the Canadian government. That grant was obtained. It was a very significant grant that allowed us to hire semi-permanent staff to run the project. The project had a staffing level of half a dozen people for several years. The Canadian government funded this project at a very significant level. I think that NSERC, which was the Canadian organization that funded the project, is still happy about the project. It was really one of the success stories of Canadian computer science.

HAIGH: You had mentioned yourself as someone working in the database area.

GONNET: Right.

HAIGH: Was this stemming from your interest in search and hashing (as discussed in the context of Maple)?

GONNET: Correct. My interest in analysis of algorithms in the end was centered around searching algorithms. At some point in time I moved a little bit towards text searching algorithms because of a Ph.D. student of mine (Ricardo Baeza-Yates) was working in the area of text

searching. He did very good work, and then later published several papers together in the area of text searching. That moved my focus of attention to databases. In particular, not just to general databases, more precisely to text searching. In some sense there were two lines in my research: one line that comes from almost the beginning analysis of algorithms and in particular searching algorithms from my Ph.D. thesis onwards; and starting in 1980, the computer algebra line that developed Maple and had several pieces of research in computer algebra.

HAIGH: From your publications in the early ‘80s, are there any that you think there were particularly significant in this area?

GONNET: In the area of text searching?

HAIGH: Or while we’re here, any of the publications from the early and mid-‘80s.

GONNET: I don't know. I am particularly happy about one paper in here. I’m not sure that the community would agree with me, but “The Analysis of Linear Probing Sort by the Use of a New Mathematical Transform,” which is coauthored by Ian Munro, is one paper that I always liked very much. In the end, this follow-up work on this paper ended up in another Ph.D. thesis of a student of Ian Munro. [“The Analysis of Linear Probing Sort by the Use of a New Mathematical Transform,” *Journal of Algorithms* 5, 1980, 451-470 with J.I. Munro].

I guess that I should be quite proud of my thesis work, which is the interpolation sequential search algorithm.

There should be another one, “An algorithmic and complex analysis of interpolation search, which I coauthored with my supervisor named Laurence Rogers. [“An Algorithmic and Complexity Analysis of Interpolation Search,” *Acta Informatica*, 31:1, January, 1980, 39-46. With L.D. Rogers and J.A. George].

HAIGH: What makes you particularly proud of those papers?

GONNET: I should be proud of my Ph.D. work, right? [Chuckles] It was a nice result. A little bit surprising. Actually a result that several people found almost simultaneously. In particular, Andrew and Francis Yao found the same result on interpolation search as I did. I don't know who found it first. We found it independently. To me it was a great concern because if you are a Ph.D. student sort of sitting on a new result and this new result is discovered by somebody else, you may risk losing your thesis. The analysis using this new mathematical transform, I always thought the mathematical transform was a very powerful tool for the analysis of these algorithms. The reasons for that are very technical and very close to the analysis of algorithms community. As I said, the fact that it produced another Ph.D. thesis is probably an indication of its value. I think it’s interesting to see that as we go along here, we are in all of the early ‘80s, it’s analysis of algorithms, searching analysis of algorithms, hashing, analysis, analysis, trees, which is searching trees. This is really mathematical, but not computer algebra. I keep on going in time. I think that the first publication on my cv that is in symbolic computation is in 1986. It’s an article that appeared on symbolic mathematical computation. I guess that the Eurocal paper of 1983 is not in my CV for some reason.

HAIGH: It may be in a separate section.

GONNET: It may have not been refereed. I would call that one a soft paper. It’s not a research paper, the paper that appeared in CACM. The real first paper that contained research results in computer algebra for myself is this “GCDHEU: Heuristic Polynomial GCD Algorithm Based on

Integer GCD Computation" that appeared in final form in 1987.[2] ["GCDHEU: Heuristic Polynomial GCD Algorithm Based on Integer GCD Computation," Journal of Symbolic Computation 7, 1989, 31-48].

You can see what I also mentioned earlier. By 1987 I had spent thousands of hours developing Maple. My CV shows a tremendous bias towards analysis of algorithms throughout that time. Actually, that corresponds also to the reality. All my promotions, all my grants, my tenure, and everything came out of the published work in analysis of algorithms, not from my work on symbolic computation. That's a question for which I don't have an answer on how to improve it, but there is a clear message here to everybody: if you are an academic researcher in a university, you should not develop software because software is going to consume a huge amount of time and is not going to give you any merits. Maybe it's a sad statement, but it's the reality. Somehow I could afford to develop Maple because I had this other stream of publications. If I had just developed Maple, I would not have been promoted and fired after several years of being assistant professor, or not renewed, or not tenured.

HAIGH: Do you think that situation would have got better or worse in the 25 years since then? Would it be easier or harder today for someone to develop a major piece of software and hope to get tenure?

GONNET: No, I think it's as hard as ever. Maybe some people would be ready to recognize it. But people would be ready to recognize that a piece of software is significant after it has proven itself, which is probably ten years after it has been produced, whereas a paper has much more immediate compensation. A paper gets accepted, gets presented in a conference or gets published in a journal. The fact that it gets published is already an indication. It gets citations. People start knowing about it. It's immediate compensation, in a very short cycle compared to software.

HAIGH: It was in 1984 that your book *Handbook of Algorithms and Data Structures* appeared. [Addison-Wesley International, London, 1984].

GONNET: Right.

HAIGH: Can you talk about the motivation behind that book?

GONNET: That book summarized several small results and several experiences that I had with algorithms. I wanted to have a handbook where if I needed to use an algorithm, I could reliably go and get some code and it would run, it would be completely tested, and it would be efficient. Also a place where I would collect all the information that is known about these algorithms, where/when they are applicable, etc. That was the main goal of the *Handbook of Algorithms and Data Structures*. I think that that goal was achieved. The book made quite a bit of an impact at the time. It was a tremendous amount of work collecting all the material for the book, running all the programs, the simulations, the guarantees that all the algorithms were correct was really a major enterprise. I spent so much time that my comment about the book is that even though it's sold in reasonable quantities, if I had worked at McDonalds I would have made more money because of the huge number of hours that I put in the book. I am quite proud of that book, and the book had a second edition with one of my students, Ricardo Baeza-Yates. He became a coauthor of a second edition of the book because he had made a substantial contribution to it. But

[2] Gonnet's cv lacks final publication details for this paper. The publication in Journal of Symbolic computation took place in 1989, not 1987. However, citations exist to earlier versions of the paper including a reference in DBLP to Bruce W. Char, Keith O. Geddes, Gaston H. Gonnet: GCDHEU: Heuristic Polynomial GCD Algorithm Based on Integer GCD Computation. *EUROSAM 1984*: 285-296.

even for the second edition I was just not able to put the amount of time that I had put for the first edition. It's an enterprise that I just cannot afford the time again to do it. I could do it at the cost of thousands of hours that I stole from my family somehow and holidays that I did not take and weekends that I did not use for anything other than the book.

HAIGH: Was it adopted as a textbook?

GONNET: No, it's not really a textbook. It's a handbook. It's for people that want to know how some particular algorithm should be implemented or want to find out what is known about certain algorithms. They can go and find all the information there. It's not a textbook.

HAIGH: How would you distinguish it from other survey books on algorithms, like Knuth's famous series?

GONNET: I don't think I can compare them. Don Knuth's books settle most of the open questions on algorithms and provide an incredibly precise framework for studying all the algorithms. They are filled with new results and a very careful collection of information. Knuth has produced textbooks or books on the subject. My idea was to have a handbook really—a place where you want to know how to code binary search trees, for example. You find the code, the code will work, and in a very summarized way you have all the complexity measures. You have a list of citations that refer to the main papers, and that's it. There are not a lot of words in my book in some sense, in the handbook. It's a reference. There's the difference between a book on a subject and a reference work on the subject.

HAIGH: It would be less comprehensive, more summarized, more aimed at someone with a practical immediate need to implement a particular algorithm?

GONNET: Right.

HAIGH: Unless there are any other aspects of your academic career during the 1980s that you would like to comment on, I suggest we turn our attention back to Maple.

GONNET: Sure.

HAIGH: One question coming out of what you said earlier would be the question of the influence of work in artificial intelligence on the system. I know that MACSYMA was more directly associated with the AI community. By the point that you were working on Maple, would you say that there was still an active interaction with AI, or were you just taking some techniques that might have been pioneered in that community and implementing them?

GONNET: I think that that was a point of departure, really, and a point of difference between Maple and the rest of the systems. None of us were from the artificial intelligence community. As a matter of fact, some of us did not think very highly of the artificial intelligence community. All of us working in the group were of the opinion that computer algebra had matured enough that it had developed its own algorithms and its own theory and its own theorems. It didn't profit any longer from the general methods of artificial intelligence, from the general search techniques, and so on. That was really a break—a break that I don't claim that we did alone; a break that many other people did at the same time.

In some sense, computer algebra had graduated out of artificial intelligence. The general techniques of artificial intelligence that were useful to solve the problems in computer algebra were too weak compared to the theorems that we could now develop, the algorithms that we could now develop and so on. It was not any longer a matter of searching blindly in a space,

trying to hit the answer. It was sort of directed search, algorithms that tell you how to find a result, whether a result exists or does not exist, and so on. From that point of view, and from looking at the people that were working in the project, I cannot think of anybody who worked in the group who had a strong background in artificial intelligence and of whom we could say, "Oh yes, this person contributed substantially to bring artificial intelligence to Maple." Almost the opposite. All of us were algorithmically oriented and not coming from artificial intelligence. To some extent that was definitely a break away from the line that MACSYMA had started.

HAIGH: Can you talk about the initial transition in terms of distribution and support? Maple stopped being something that people in the group would have been manually copying onto tape and mailing out to people and gained a new relationship with Watcom where it was distributed outside the group itself?

GONNET: Sure. We recognized very early that at some point the tasks of distribution could not be done by the research group for various reasons. The tasks were administrative. Users wanted to have a phone where they could call and get their questions answered. Users wanted a manual. Users wanted all types of documentation, a nice box maybe, and so on. We could not provide any of that.

The natural solution for this was to find somebody who would be interested in doing all these jobs for a percentage of the revenue. Watcom was the right vehicle at the time. It was a company formed exactly for that purpose, to distribute Waterloo's Fortran compilers and other software that had been generated by the same group. The chairman (and probably held some other positions) was Wes Graham. Wes, of course, was a colleague of ours and knew about the project and had been involved in the very initial meetings of the group. He was always in touch, and was always ready to distribute Maple. He also saw the acceptance that Maple was having in the community, so he was interested in having Watcom distribute the software.

For the group it was a major effort to get a version that was acceptable to Watcom for distribution. It was a self-imposed major effort, but also a requirement from Watcom. Watcom wanted a product that will have certain characteristics, and we wanted to compete in certain areas, and say, "Well, if we are going to have a commercial product, we have to do this, we have to do this, we have to do that." That ended up being an extremely painful process which made me make a decision that I applied since then all the time, which is whenever you are going to release a version, you release a version based on time not based on features. You say, "Okay, next month or the first of May we release a version with whatever we have in there." If something didn't make it, that's too bad; it will make it in the next version. As opposed to saying that for the next version, "we want to do this and this and this and this." Some things are going to be done quickly, some things are going to be done at some normal speed, and some things are going to take forever to finish. You delay your new version by the worst case. The thing that takes the longest is the one that delays your version. In some cases this takes forever because as you go along, people want to put more things in the version and the goals keep on changing and it's a nightmare. So I learned a very important lesson there: you release by time not by features, and thereafter we released by time not by features. Even the company nowadays, Maplesoft, is basically releasing by time not by features. Those lessons are sometimes very difficult to learn.

I have to say that my role in the group at this time was a little bit of, I wouldn't say the police, but every group has to have a person who, when something needs to be done and solved and fixed, is the one person who will eventually do it. If you don't have such a person, if the buck doesn't stop anyplace, then you don't have a software project. No matter how difficult the

problem is, or how tricky the solution is, someone has to at some point say, “Fine. I’m going to solve it.” And has the ability of solving and has the time of solving it, and will also make all the decisions. “Okay, today we froze the version. This is it. No more changes. I am cutting it. I am making the tapes. I am branching the new version.” In the case of Maple, that was me. Of course that made me extremely unpopular quite often because I would make decision like, “Okay, no more changes after today. This is it. We have a new version.” Or, “This has to be rolled back.” Or whatever.

On the other hand, that also means that when there is a serious bug, and I can remember some in garbage collection that were extremely unpleasant because they show up much later and in a very damaging way, somebody has to go and fix them. Sometimes it’s a lot of fun to code something that does something very interesting. I can assure you it’s not a lot of fun to fix some obscure bug in garbage collection. Nobody wants to fix that. I don’t want to fix that. Nobody wants to fix that. For the goodness of the project, you have to balance the good and the bad. The bad is you’ll have to spend a huge number of hours fixing something that is boring, tedious, difficult to find. On the good side the project goes on, and if the project doesn’t have some level of quality, it will not go on.

[Tape 3 of 6, Side B]

HAIGH: I imagine that kind of discipline would be relatively hard to achieve in an academic group rather than in a traditional corporate setting.

GONNET: Correct. In a corporate setting you say, “The buck stops here. You are paid a salary. This is your job. You have to do it. If you don’t fix this many bugs, or if you don’t do it, I’ll fire you.” In an academic environment you cannot do it. It has to be one of the main responsible persons of the project that takes the role of doing it. Otherwise it just doesn’t happen. I guess that that’s the difference between projects that really produce software or projects that produce prototypes that are nice programs that sometimes work, sometimes don’t. That’s really the main difference.

HAIGH: I think earlier you had alluded to things like documentation and packaging.

GONNET: Yes.

HAIGH: Were the changes needed for the distributed version more fundamental than that, in terms of the program itself?

GONNET: Yes, there were some demands about graphical output and portability and documentation. We worked very hard producing tutorials and manuals for the language. This was a different time. Documentation was normally printed in a book. At the same time, we had to make arrangements to print a book. I don’t remember exactly how the first book was printed, but I think it was Watcom that printed the first versions of the book. Of course we had to provide all the material; they would just do the printing. Getting a manual done is a huge amount of effort. It’s like getting a book done, right? Keith Geddes had the main responsibility, and Bruce Char to some extent too, the main responsibility on the documentation. Then every box goes with the software on a tape or whatever media and a book—that becomes the package. The design of the package was Watcom’s responsibility, and we don’t have anything to do on that end.

I was more responsible on the side of the software. The kernel was 100% my responsibility, and the Maple libraries were by and large everybody's. I was playing quite a bit the role of policeman in the Maple library and the kernel.

HAIGH: When Watcom distributed the software, was this thought of as an arrangement that would just cover the costs of distribution and support, or was it intended to bring revenues back in order to support further work?

GONNET: It was a very generous arrangement for the University. It was an arrangement that was based on a formula which gave 15% to the authors, and the remaining 85% was divided between the university (the Symbolic Computation Group) and Watcom as the group that was doing the distribution. It was very generous to the university and very generous to the authors. This was actually something that was the common denominator in Watcom and in previous experiences of this group of people to have this distribution of revenue. Watcom quickly realized that they were not making too much money distributing Maple. They wanted to make a bigger investment in marketing and packaging and so on, but with only 42% of the revenues they could not do it. We wouldn't move on the 42%. We were giving Watcom a finalized product, right? We were not saying, "Okay. You develop the software. We just collect the royalties." We were giving them the final software. We had programmers, we had students that we were paying to support the product and to write libraries and to improve the code, so we had good reasons for saying, "Our 42% is completely used." By the way, the authors not collecting any royalties at the time. The author royalties were used to support the group. So we had good reasons to keep the arrangement that we had. Eventually this arrangement was sort of a dead-end street for Watcom and at some point we decided to part ways.

HAIGH: How much had the software been selling for under this arrangement?

GONNET: I don't remember, but I think it was probably around $500-$1,000 depending on the type of computer. The price was very sensitive to how fast the computer was and how many users could be using Maple. As I mentioned before, we had the square root law for the price, so one license was one unit; n licenses were the square root of n times the base price, so nine licenses will pay three times the basic price.

HAIGH: That arrangement had originated with Watcom?

GONNET: No, that was my idea. The square root law was my idea, if I remember correctly.

HAIGH: Did the user base continue to consist primarily of teaching with some use by academic researchers?

GONNET: Yes. At the time it was, I wouldn't say 100%, but as close as you can get to 100% of academic use, either people that were teaching or researchers in academia that were using Maple for their research. We had some notable exceptions. Bell Labs through an arrangement with a colleague of mine, Andrew Odlyzko had a copy of the source of Maple, which was quite privileged. This was very useful for us because Bell Labs at the time was still very dominant developing Unix and C compilers. They would test their compilers by compiling Maple at the time. It was very convenient for us to have a version in Bell Labs with the source and everything. They were using it for whatever they wanted, but we were also getting our value in the sense that we knew that compilers produced by Bell Labs would compile Maple.

HAIGH: Did any significant non-academic niches develop for Maple during the 1980s?

GONNET: I don't remember anyone in particular. As I said, most were academic. The history in general of Maple has been a history of having a very wide acceptance in the academic environment. All the companies that distributed Maple have a history of trying to break into the commercial market, being unsuccessful in the commercial market, spending huge amounts of money and effort in trying to break into the commercial market, and never getting enough sales to justify the expense. And in certain cases even disregarding the academic market, which in the end was the one that was paying all the bills and making the company profitable.

HAIGH: Why do you think it was largely unsuccessful in trying to break into the commercial market?

GONNET: Well, it was not unsuccessful. The relation between revenue and effort or revenue and expenses was much more favorable to the academic market than to the commercial market. It would take huge campaigns and lots of effort and marketing and so on and meeting lots of demands to sell to the commercial market, and with half of the effort or a fourth of the effort we would sell much more to the academic market.

HAIGH: So you think that was more a result of the amount of money that would have to have been spent to achieve success in the commercial market than of anything inherent in the product itself?

GONNET: I think there were some things that could be attributed to the product, no question. While it was in the University Maple was respected for its mathematics and in its efficiency, not for form of the output. This was something that was definitely a miscalculation of all of us who were/are always worried about having correct results and having results fast and efficiently and with little memory. We never thought of a fancy, colorful, graph as being important. Our internal joke was: "what would you prefer, a correct result or an incorrect result with fancy colors?" Our big surprise was that people preferred an incorrect result with fancy colors, when we saw the success of Mathematica, which really was an unreliable system, but it had much more advanced input/output than Maple did. I guess that the fact was that we were more concerned with the quality of the algorithms, the quality of the mathematics, the efficiency of the system and so on make us disregard the aspects of how easy it is to input, how easy it is to display results and so on, which may be more relevant for a commercial environments where somebody has to show the results or somebody will have more than one product and will have the option. Whenever they have the option, they will take the option that is the most pleasing to work. If anything, in the battle with Mathematica, our biggest mistake was that one, was to concentrate on efficiency and correctness in algorithms and coverage, as opposed to how easy it is to input information into the system and how pretty and convenient is the output.

HAIGH: I'll return to that question of commercial competitiveness a little later. I just had one more question on Watcom. I think yesterday you had said that the Maple project had originally been envisaged in part as an attempt to produce a WATFOR of computational algebra. One obvious aspect of that would be that it would be fast and efficient. Do you think that there's any kind of philosophical or technical relationship between the projects that runs deeper than that?

GONNET: No. In some sense it's a complete coincidence that WATFOR and Maple were developed at the same university. Yes, there is a paper that was saying it would be nice if Waterloo would develop the equivalent of WATFOR for computer algebra. Yes, we developed it, and yes, it was marketed at some point by Watcom, which is the same company that was marketing WATFOR and WATFIV, but the intersections of the groups was empty. The

philosophies of the groups were quite disjoint too, so we had no exchange of people. We were originally based on Honeywell GECOS, and then very quickly UNIX. These Watfor/fiv groups were mostly IBM oriented. Although we were in the same department and in the same university (and we even had Wes Graham who was very active in Watcom and helped us at some point, or met with us, in the forming of Maple) in the end there was really no influence from one group to the other. Well, you could say Waterloo has a tradition of good software development, but it cannot go beyond a general statement like that.

HAIGH: Would you like to talk now then about the creation of Waterloo Maple Inc?

GONNET: It's actually WMSI, Waterloo Maple Software Inc., although it may have had different names at different times. Now it's called Maplesoft for sure. The Waterloo Maple Software name has been abandoned.

The distribution was done by Watcom starting in 1984 and 1985, and as I mentioned earlier, at some point Watcom wasn't happy with the royalty arrangements. They wanted to do more; they couldn't do more. By not having a bigger cut, they were not motivated to sell more or to invest more. At some point they said, "We don't want to do this any longer unless we take control." Basically we said, "We're not going to give you control." At some point you have to sort of take a risk and invest some real money, start paying some salaries, start a corporation, and hope that you are not going to lose your shirt. That's basically what we did. Keith Geddes and myself, we both invested some small amount of money—very large for us, but small amount of money in terms of real money—and other members of the group also invested, and we all became shareholders of this new company. We were rolling. We rented some office space in North Waterloo. Very soon we had four employees, if I remember correctly. Then we went in through quite rapid growth, I would say, because the software was selling already. What we needed to do was to collect money basically. It was a very privileged situation.

HAIGH: Was this 1988?

GONNET: I don't remember exactly when it was, but it's probably around 1988, yes. Around that time frame.

HAIGH: Had the arrangement with Watcom been bringing in enough money to support the group of people who were developing Maple?

GONNET: The arrangement was very generous. It was bringing substantial revenue to the group to maintain quite a bit of development going on. The university was also very generous because in theory all this revenue, this 42.5% should have gone to the University, but the University was channeling it all to the Symbolic Computation Group. So we were using all of it to pay salaries. I don't remember the exact numbers, but we probably had around ten staff members paid, between grad students and fulltime staff that were working in the group.

HAIGH: Did the creation of the company mean the complete transfer of Maple work away from the University?

GONNET: No, not at all. At first the company was viewed as a replacement for Watcom with the idea that it will eventually take more and more responsibilities out of some parts of the development that were clearly not suitable to be done in the University. The first thing that came to mind was that the SCG should not be worried about porting to all these different hardware platforms. The SCG was not well equipped to write fancy user interfaces. The SCG should concentrate on writing good algorithms and developing good algorithms. It was probably not my

idea that the SCG would pull out completely, but eventually history showed that the company took leadership and took over all the activities, the research, and the other software development.

HAIGH: Do you know what year that would have been in which the transition was complete and the University was no longer significantly involved?

GONNET: It has been gradual. I think it went on through the 1990s with the University slowly losing responsibilities. In 1989 I left Waterloo, and Mike Monagan also left Waterloo, so that was a significant shift. The Symbolic Computation Group was less active just simply because it had less people working in it. At the same time, the company kept on growing, I started a group here in Zurich that was quite active, but it's not the same to be in Zurich as to be in Waterloo. Around 1994 the company got into difficulties in management and started cutting the ties with the University and with their research groups and renegotiating the contracts for distribution. Then the company took complete leadership in the direction of the product.

HAIGH: Prior to that, how much money had been coming back to the University group from sales of Maple? Was it a similar amount that you had been getting from Watcom?

GONNET: I really don't remember the exact numbers. It was a generous amount. It was generous enough as to maintain all the activities that we wanted to maintain in the Symbolic Computation Group.

HAIGH: You mentioned a small early grant you received for the Maple work. Did other outside funding follow?

GONNET: We had other smaller grants, and we had our own operating grants from our NSERC funding agency. I don't remember any significant funding grant outside of those initial funds.

HAIGH: Most of the funding would have come from revenues than even from an early stage?

GONNET: Right.

HAIGH: Can you talk about developments in Maple itself during this period?

GONNET: I can't pinpoint anything specifically. The system was being developed version after version. I guess a significant point was when I left Waterloo, and at the same time the company hired Ron Neumann as the CEO. Ron was a very dynamic person. Saw the future very clearly, and saw the niche that Maple was in, very clearly, and was a very dedicated worker. The company really took off with him. There was actually a very positive interaction between Ron Neumann and myself. I was traveling to Waterloo not that often but often enough, once every two or three months, and we used to communicate very, very effectively. My role at the time was basically to bring lots of ideas, and Ron's job was to filter those ideas, pick up the ones that were good enough, and implement them.

This was working very, very well for the company and for Maple. So much that Maple grew at a fantastic rate. If we take something like five or seven years of Ron Neumann's leadership (I was the president, he was the CEO), the company grew at a composed 100% per year for five or seven years. You can grow for 100% per year for a limited amount of time, but to grow it for perhaps seven years all (five years for sure), this is very significant. This is not just a fluke. This is not just playing with the numbers from one year to the next. This growth had some substance. This growth was based on real activities, really good ideas that were being produced and implemented. Just to mention a few, at some point we decided on a university wide policy. We were going to give university-wide licenses. That was considered crazy by some people, but it

was a great success. Lot's of universities got site licenses. Maybe we would be able to sell more individually, but universities loved this. Every year a big check was coming from those universities. That was excellent. It also pushed Maple very prominently. Very early in the game another idea was to port to the PCs. At the time the PCs were not very prominent and were not really the main line of academic institutions. It was viewed as a bad idea. We decided we will port to the PCs. It was a great success. Again, another incremental where we grew the market share tremendously.

HAIGH: Can you remember what year the PC version would have appeared?

GONNET: No, I don't remember exactly, but it must have been very, very early 1990s. I remember one of my students working on a prototype for the PC, and it must have been 1991 that the PC version came out.[3] At some other incremental step was making a deal with a company in Massachusetts called Mathsoft. Mathsoft was producing a piece of numerical software. Very user-friendly, which I forget what it was called at the time.

HAIGH: Mathcad perhaps?

GONNET: Mathcad. Great. You're absolutely right. Which had little to do with CAD, but it was called Mathcad.

HAIGH: I remember it looked like a word processor.

GONNET: Right. It looked like a word processor except that the numbers could be tied by formulas and the numbers were alive and you could do computations with those numbers. Very handy for any engineer that wants to present a description of some process or something. The deal was that we would incorporate Maple in Mathcad so that you could now do symbolic computations in Mathcad and not only just numerical computation. For them it was a great deal because at the time their space was getting a little bit crowded with a lot of people that were getting into that same space, and they took a tremendous advantage from everybody else by having computer algebra. A real advantage. They dominated the market for some time. For us it was a tremendous amount of revenue. It just was a quantum leap.

And so Ron Neumann was very dedicated, was a person that would work 20 hours per day or something like that. Very dynamic. We were making an excellent team in working together. Things were happening. You don't get 100% growth per year just on doing the same thing everyday that you did before, doing it just a little bit better. You have to have quantum leaps of new ideas that bring you business, or new forms of doing business. At that time, Mathematica was already out, probably, so we had realized that there was a big threat. All of a sudden we were not the only kids on the block. Because we were very complacent in the late 1980s, we were the only choice for most people, so why bother? Why do better input/output? You have to use us. If you want to do computer algebra you have to use Maple. Why bother?

HAIGH: So at that point the input and output were all plain ASCII?

GONNET: Yes, it was ASCII. For Maple it was ASCII all the time. By and large, for most people input is still ASCII. For the output, you don't want it to be ASCII. I would say that if you are an unsophisticated user, you may want to use the input with standard math. But if you really want to use the input many times you will hate the slowness of using the mouse in a graphical

[3] Reports on the internet suggest that Maple 4.3 ran on 80386 PC compatible computers as early as 1988, in the initial Waterloo Maple Software release. This has not been further verified.
http://www.math.utsa.edu/mirrors/maple/mplhist.htm

input. You really want to type in things, in ASCII. That's my view at least at this point. Maybe smart tablets will change this, but I am not completely sure.

At some other point Maple entered into an agreement and acquired a company owned by Alan Bonadio that had two products. One was called Theorist, which was a Macintosh product. I wouldn't call it a competitor of Maple. It was a computer algebra system in the end, but with a definite orientation towards teaching and towards proving simple things. Alan is a great guy, and had a great product. In the end it did not work well. Somehow I think it was Maple's failure to integrate the product properly into the line. In the end that gave us some growth, but it produced a lot of unhappiness on both sides.

HAIGH: So that remained as a separate product?

GONNET: That remained as a separate product. Alan Bonadio had two products. The other one was basically a product to display mathematics in a fancy way. Expressionist. Expressionist was just a way of incorporating math into your general document. Sort of fancy math. If you need to display a tensor or an integral in a complicated space, if you need all sorts of symbols up and down, and funny fonts and so on, that would be the right product.

These products were very much Macintosh oriented. The Macintosh market was not the biggest market for Maple originally because the market for Maple was definitely the UNIX market. Then it slowly became the PC market. As a matter of fact, quite often Maple did not have good versions on the Macintosh. The versions in the Macintosh looked a little bit clunky for the Macintosh.

To some extent I may be a little bit too dim on the relationship with Theorist. It was a very good idea. Originally it created a lot of sales and revenue and market share for Maple. It was just that it was not managed properly. In the end it was not a happy relationship for everybody involved.

HAIGH: How did your own personal involvement with Maple change over this period?

GONNET: There was a dramatic change in 1994. Before 1994 my involvement with Maple was as I described it. I left Waterloo without much animosity, but I had a significant group in Zurich. Most of the people in my group were doing computer algebra at the time. We had a particular arrangement with the Maple company that was funding a little bit of our research in Zurich. Not in a very significant way, but it was funding our research. My relationship with the company was mostly through Ron Neumann. As I said, it was a very positive type of cooperation there.

HAIGH: As you described it earlier, during the Watcom period you had been serving as the main project manager in Maple, and you had also said that you had been personally responsible for work on the kernel. Did that change immediately with the move to ETH?

GONNET: No, no, no. By necessity, I kept on being responsible for the kernel for many years, even once I had moved here to Switzerland.

HAIGH: Were you working on that with the group here that you just mentioned?

GONNET: Yes. When I came to Switzerland, I was offered to bring several people. I brought with me Mike Monagan that was doing computer algebra, and Tim Snider, who was doing OED related stuff. I also made an offer to Tim Bray, who was also doing dictionary at the time, and unfortunately, he decided not to come, which was a pity. I had a very good working relation with Tim Bray. Tim Bray then when on to design XML and stuff like that. Tim Bray was our project manager for the OED project. Actually, XML is the result of us (in the OED project) being so

unhappy with SGML. The OED project was probably one of the biggest early users of SGML. We were extremely unhappy at how clumsy SGML's definitions were. We always said, "We need a better SGML. We need a simpler SGML." Tim Bray went and did it. That's called XML.

So very quickly I had five or six grad students and post-docs working in my group. Most of them (except for Tim Snider) doing computer algebra work, really Maple. I kept on working in Maple very closely for, I would say, the first five years in Switzerland between 1989 and '94.

Session 3 begins, the afternoon of March 17, 2005.

HAIGH: I think before we broke you implied that there was some change in your relationship with the company in 1994.

GONNET: Right. It is a sad part of the history of Maple in some sense. It was a series of events that had to do with my role in Maple. I mentioned my role of policeman, and I said that this was causing friction very often, and it's true. Every time I was making a decision (even though people most of the time would recognize that decisions had to be taken and were not bad decisions) it was seen as an authoritative mode that I was playing, that I was being a dictator. That continued and got worse when I was at a distance because I was making decisions from a distance about the product and about the direction that things should go. "Why insist on making decisions from a distance? Why is he making these decisions at all?" That created always a little bit of friction. There were a series of events that produced a complete break between Zurich and Waterloo. This started with the company. As I said, the company was growing every year by 100% for many years.

HAIGH: Do you know what size this had reached by 1994 in terms of employees?

GONNET: Yes, about 70 employees, so it was getting to be a reasonable size.

However, there was a little bit of a problem with Ron Neumann. Ron Neumann had been excellent to run the company all the time, but had a bit of a problem in delegating. He had not created the intermediate levels of management that he could trust and rely to keep on growing the company. In other words, he was managing basically everything directly. When it was ten people, it was perfect. When it was 20 people, no sweat. When it was 30 people, it was getting to be a problem. He was starting to work 15 hours a day. When it was 70 people he was just not able to run everything. He was killing himself because he was working a tremendous number of hours. Things were not done, projects were falling by the side, and the company was showing some problems of bad management. At the time I couldn't see that, but in retrospect it was exactly that problem. He was a perfect manager as long as he was managing people directly. When it got to be 70 people, you cannot possibly manage 70 people directly. You have to create intermediate structure. He did not create the intermediate structures or the structures were created but were not effective and/or autonomous.

So the straw that broke the camel's back was when we went from having our most profitable month ever, to running out of money in such a way that we almost had to close the operations. It was just odd. A couple things didn't happen, and a loan that was supposed to happen didn't come. Then the bank pulled the plug because we were in the red instead of being in the black. All of a sudden it was a disaster area. I said, "It cannot be that we have survived for so many years, that November was our most profitable month ever, and in January we're thinking that we cannot make the payroll. It's just not thinkable. It's not correct." Ron Neumann reacted very poorly to this, and I probably also reacted very poorly, and we decided on a change of management. At that time there was a person that was running the European operation in

Heidelberg called Dieter Hensler. Hensler appeared to be a very competent manager. He actually motivated most of this change. He obviously wanted to position himself as the new leader of the company. I sort of believed that that change was a positive change.

So I started this change of management. I had all the power to terminate Ron's position, and I did. But this created an incredible havoc within the rest of the shareholders who accused me of being heavy-handed and so on. Of course the rest of the shareholders were not worried about paying the payroll or whether the company was going to survive or not. They were too remote to realize what was the situation in the company every day. Not close enough to the reality to know what was happening.

[Tape 4 of 6, Side A]

GONNET: Dieter Hensler became the next president of the company, but there was irreparable damage done to the relationship between myself and the rest of the board. A long, stupid, and damaging battle ensued. From there on it was all negative, I have to say. I lost all respect that I had for my colleagues. At some point they forced me out of the board, even though I was the largest shareholder. At some point I was so disgusted that, with time, I decided to cut all the ties with the company. So in the end it was a sad story. All the time that I spent for the company, all the effort that I spent….

Well, in some sense I was spared. I got my shares, I got my value, I got my author rights, and so on. These people maybe they thought that I was doing the wrong thing. Maybe I was doing the wrong thing, but at least I was doing something. At least I had brought the company to the state that it was. Maybe they thought that they could do better. Maybe they saw their chance of being a hero in some circumstance. I don't know what motivated them, but they were very stupid, very selfish, very damaging in the end. I actually have nothing good to say about them in the later part of the company. I have no use for their time, no use for their actions whatsoever.

HAIGH: At that point would the board still have been made up of the other founders?

GONNET: The first step was to form a board that was larger and included some external people. The board was made up of Keith Geddes, myself, one nominee of Keith, one nominee of myself, and I think Dieter Hensler was the other one. My nominee was Wes Graham, who had been somewhat involved and we had a reasonable relation. Unfortunately I have nothing good to say about him. It turned out that his sole purpose on the Maple board was to find things that he could blame on me (even though he was my nominee). I don't know what it was, if it was professional jealously or what, but again that's a person that I have nothing good to say about. Although in the earlier stages he was a respectable colleague; in the later stages I changed my mind drastically. Sadly enough he lost any respectability left with me and with the people that knew this affair, a sad way of ending his career.

I don't know the real story of how it developed. The situation was that Dieter Hensler made some alliance in particular with Wes Graham, because Dieter Hensler could not manage the company either. When I realized that things were not going well I was trying to get the company back on its feet and working again. Dieter Hensler who was the person that I placed as CEO at the time of the previous crisis really stabbed me in the back in the worst possible way, and sort of became a friend of all the others and basically used the momentum against me to his advantage. That was the time that I said, "Okay. I am parting." At some point in that process I managed to get funding to acquire the company from Springer-Verlag.

HAIGH: Springer-Verlag supplied the money, or they owned the company?

GONNET: No. Springer-Verlag, through a contact of mine, was ready to invest enough money to buy all the shares to control Waterloo Maple.

HAIGH: At that point were the shares all still in the hands of the founders, or had additional investors been found?

GONNET: Mostly in the hands of the founders. There were several people that wanted to sell their shares. In a private corporation if you find a buyer for your shares, nobody is going to block it. Waterloo Maple has the dubious privilege of having blocked the sale of shares of its shareholders to prevent me from getting control of the company. It's a rare and unique merit that they have, having passed board resolutions preventing shareholders from selling their shares to me. They got to that level of pettiness. After such events I said, "Get lost. You don't deserve my attention any longer." That's basically what has happened.

They have stumbled ever since, and they have lost the battle with Mathematica. There haven't been any new ideas in the company since 1994. Before 1994, every year we could say, "Oh, this year we had doubled the market share by doing this. This year we had doubled the market share by doing that." Since then it has been nothing. They are at the same level of staff virtually, around 100. Same amount of revenue, and the market share shrinks every year. Mathematica beats them in all aspects unfortunately, but that's the sad story. But it's sad for me because I still like Maple and I still like the product. But it's a consequence of a board of incompetent people running the company. It's not that the product was not good enough or that something else happened. In this case it's the consequence of the actions. If you have incompetent people making incompetent decisions, that's what you're going to have. That's the history in a sense. The whole breakup was not instant. It started in 1994. It lasted for a couple of years. It was only about three or four years ago that I eventually sold everything that I had. I sold my shares, I sold my rights, and so on.

HAIGH: That would have been around 2001?

GONNET: I think that it finished in around 2001. I was the largest shareholder. Still, I couldn't do anything in the company. And the largest shareholder by a significant amount. I owned about 30% of the shares of the company, and still couldn't do anything because people were convinced that they should vote in block for whatever. It was plain stupidity, but human stupidity is rampant in this world.

In summary, I have a lot of good things to say about my colleagues in the constructive phase when we were building the product. It was great fun, and we had great cooperation. We did something good. We constructed something out of nothing. We had an excellent team spirit to build a product. Of course we were discussing all the time or exchanging ideas. We were not happy continuously, but we were building something in a positive way. After 1994, this became just a petty group of people that were just grabbing for some power and not seeing the big picture. That's where they are going. They're going nowhere.

HAIGH: I have a few questions about development of the software itself during the late '80s and early '90s. One of them would be that you had mentioned earlier that something like a third or a quarter of the library code was completed by users in the early years. Did that pattern of user contributions continue when the software was commercially distributed?

GONNET: Unfortunately not. The company probably, correctly so, started cleaning up the code, and research groups continued to maintain some of the code. There are many, many parts of

Maple that are still maintained by some researchers. I was maintaining some of my code until as recent as three or four years ago.

But it's a very tricky situation when you have very good code that has been contributed by someone who then disappears or is completely uninterested in maintaining this code. The code may be excellent code, but it's written in 1989 let's say. It's excellent for 1989. Now Maple starts evolving. The theory starts evolving/improving too. There may be better algorithms. There may be better constructions in the language that allows you to do whatever you were doing more efficiently, or you have possibilities of covering other domains that you were not covering before. So the code ages. Unless somebody is actively maintaining it, the code ages (sometimes called "bit rot"). No matter how good the code is, it will age. In the case of computer algebra and in the case of Maple, things tend to age pretty quickly. They require maintenance. Without somebody actively maintaining the code what happens is that, no matter how good the code is initially, at some point it becomes a liability. At some point it becomes a piece of code that is too slow, doesn't do the job properly, is coded using archaic constructs, inefficient constructs, and so on. At some point it is either removed, or the company or somebody else actively takes over that code. It's very difficult for any researcher to actively take over someone else's code. It's not the type of thing that researchers will normally do. If you have a new idea, you start from scratch. You are not going to go and patch something in someone else's code. It's really the job of the company to maintain that code. The company has maintained some of the code; in some cases they have rewritten it. I would say that right now most likely the company is maintaining and writing 95% of the code that comes out of Maple, which in some sense is what it should be. Researchers may produce a very good piece of code, but we cannot ensure that they are going to maintain it forever.

HAIGH: I understand that it was with Maple V that a graphical user interface and graphical output capabilities were first added.

GONNET: Right. I don't remember exactly which version it was, but it was around Maple V, yes. Maple 5 had several subversions. Now in more recent times they have gone for a new number every year.

HAIGH: Was the addition of those capabilities in response to Mathematica?

GONNET: Yes, there's no question. As I mentioned before we were very complacent, and that was our biggest mistake commercially: to be complacent that we have good mathematics and good efficiency and a good code base and that we can write algorithms that can be maintained and so on. I don't know what exactly the situation is right now in the internals of Mathematica, but for a long time most of the Mathematica libraries were written in C. Consequently they had a code base that was extremely difficult to maintain. People were telling me that because of lack of modularity, everybody who wanted to do something in Mathematica had to basically start from scratch. If you wanted to do integration, if you had to do something with polynomials, well you had to do it yourself because you could never trust the rest of the system because it was changing, because it was not reliable. You didn't want to inherit someone else's bugs.

Fortunately, that was not the case in Maple. If that was happening, it was happening to a much lesser extent. We had a better code base. We had a better language. We had a more efficient system. We were just too complacent with our position. We were thinking users will appreciate our efficiency and our correctness. They are not going to go to Mathematica, and we were

wrong. It was a shock to discover that we were wrong on that one, that users would prefer better input and output, even at the sacrifice of correctness. That was a rude awakening for us.

HAIGH: When graphical capabilities were added, do you think that they were on a level that could compete with Mathematica, or did they continue to lag?

GONNET: No, I think that they continued to lag. I think that the company didn't made a serious effort at first, and only at certain times they had the right people so that they could gain on Mathematica in relative terms with respect to their input/output and facilities for displaying results, and even facilities for acquiring data. I think it was also a goal that was not so close to the heart to the technical people at the company. It was maybe closer to the heart of management, but I'm not even sure about that. It's not a complete surprise that they always lagged behind. Maple was always lagging a little bit behind or significantly behind in input/output capabilities. If you wanted to write a live document where the math would be computed, I would say that almost always Mathematica was ahead. Almost always Mathematica would give you more facilities to add it, more facilities to do it.

Now, we go back to what I said before. The significant growth happens when you have very significant ideas that give you quantum leaps in the market. Maple had dozens of opportunities of having these quantum leaps. It's probably too easy and unfair to say now that "they should have done this and this and this and they would be ahead." But it is not completely unfair because that was done at some point. At some point we were getting ahead. At some point we were recovering market share from Mathematica or regaining the market share that we had before. That didn't continue. That didn't happen in '95 and onwards. I think that that's what pushed Maple farther and farther behind. It was the fact that, for example, the Web was never embraced. We had very reasonable ideas about OpenMath. It was never embraced. They just gave lip service to it. There were all sorts of things that were opportunities that were not pursued. Maybe not all of them were right. It's very easy to now say "we should have done this and this" now that we are in the future. I can see a project and a company between 1980 and 1994 that was doing new things, solving new problems, and in terms of marketing having new ideas, selling in different ways, and selling to different people. All of a sudden it goes into basically a survival mode where management writes nice colorful reports to a sleepy Board but does nothing new. This is a big danger in a technical company.

HAIGH: Was the Axiom package marketed by NAG something that you were aware of in the early 90s?

GONNET: Yes, we were aware. Not only we were aware. We had very close ties with the Scratchpad group that produced Axiom that then was distributed by NAG. Stephen Watt was a Ph.D. student of Keith Geddes. He was an early Maple contributor, a very substantial Maple contributor. He was the one that designed the Maple leaf with the ASCII characters that still the TTY versions will print. He went and took a prominent role in IBM. He was working in Scratchpad at the time, Axiom later. Dick Jenks was the manager of the Scratchpad Group. A very good friend of mine. We had an excellent relation all the time. We were sharing ideas. We were quite open about what we were doing. We knew quite well what they were doing.

It had always been my opinion, and it continues to be my opinion and I think that I have been proven right, that Axiom was way too complicated for almost every normal use. Scratchpad and Axiom had a remarkable object-orientation model which allowed you to define the mathematics in a very precise way and do the mathematics in a very precise and efficient way. But by the time

that you had defined it, say that you had a field over this and it was commutative and there was this and there was that, then most users would not know what they were doing and would not be able to identify in which domain they were working. The system itself became so complicated that even the experts had problems trying to get it to do what they wanted, whereas Maple had preserved this initial flavor that you sit down, you type something to it, and you can start doing useful work immediately. I think that Maple managed to lose that in the later years because I think that usability of Maple has gone down in recent times. But for the average user of computer algebra, the usability of Axiom was terrible. I think that that's the reason why Axiom was never a success and was barely distributed. I don't even know whether NAG still promotes it or has dropped it.

HAIGH: They stopped distributing it in 2001. I think they have given the rights to an open source group attempting to revive it.

GONNET: Yes, that's a possibility. Talking about open source, about three or four years ago, I mentioned this to Maple management. We had a sort of high-level discussion, and I was asked, "What would you do with Maple?" We were losing the battle with Mathematica. What should we do? I said, "Open source is your answer. Open source the programs, (the kernel, and the library) and sell the manuals, sell the consulting, sell the know-how for people that want to sell anything that is built on top on Maple. But get the open source energy…"

There is quite a bit of energy with open source. There is positive energy when people think it's neat that we know exactly what we have, even though they are not going to touch, they are not going to look at it. But you feel much safer if you run Linux that you know exactly what you are running, or you could look exactly at what you are running, as opposed when you run Windows that they may be stashing information behind your back and you don't know that it's happening. There's that effect. Then there is the effect of people that see a problem, are very worried about solving their problem, want to fix it, and actually go and fix it. Now you have a positive contribution. Sometimes it's a negative contribution because they fix something and they break something else.

But overall, there are two aspects of open software that make it more popular. Maple desperately needs to be more popular if it's going to win the battle with Mathematica. I don't think it's going to win the battle with Mathematica, sadly enough.

HAIGH: So you raised this possibility of an open source model for Maple.

GONNET: I raised the possibility. But I didn't have any influence in the company any longer, and I'm sure that some of the people may have raised it. Obviously they didn't take it. Maybe now it's even too late to go that route.

HAIGH: I'm aware that the current version of Maple has also developed something called toolboxes to target particular areas. Do you know anything about those?

GONNET: No. Well, I know the concept. The concept has been around for a long time. It has been around in the software industry in general. I guess that the first people that made this quite obvious were the people selling games. They would sell the main computer at a very low price, and then the cartridges were the games at a relatively high price. It's the way of packaging. You entice the user with a very low price for the hardware, and then you make your money out of the packages.

We experience that in a hard way with Mathcad. The Maple version was attached to the main version of Mathcad. Our royalties were a percentage of the price of Mathcad. Mathcad went exactly that route, and they drop and drop and drop the price of the main product, and they started charging more and more and more for the application packages. Consequently, causing us great harm. A percentage that was set for our product that was selling for $500, at some point it was being sold at $49, so our royalties were 1/10 of what we were expecting to see. Basically we were getting pennies or maybe dimes for a version of Maple that was going out with Mathcad. In the end, after two or three years, it ended being a bad deal.

That game of packages has been played many times. There is a marketing part of it, also you access a vertical market with a package. In a package you can allow yourself to speak the language, use the constructs, use the functions, and so of that a particular sector of your population needs. Verticals are known in many pieces of software. Matlab had these packages for a very long time. Mathematica had packages. It's time that Maple would have its packages. You need to capture the audience that requires some specific math, and a package is an excellent way of capturing that audience. You also need people that have the expertise of the area and know the jargon and know the math and know the know-how of the area. The company in the early stages did not have that, and did not pursue the verticals in a very significant way. I guess that direction was pursued in some way. We had some packages, but the packages in modern terms would be called "amateurish".

HAIGH: You had already talked a little bit about the relationship with MathWorks and Matlab to integrate symbolic capabilities into their numerical product. Was that something that was used by a significant number of people in practice?

GONNET: I really don't know. At some point we arranged for a relatively symmetric situation by which Matlab will have Maple incorporated, and we could sell Matlab in some form or other. MathWorks was very diligent in integrating Maple in a very reasonable way and providing the symbolic computation facilities through a package that was in the same sort of language that Matlab has.

We were quite happy with that at first, but it sort of backfired. It was at a time that I was getting less involved with the company. But I have seen European distributors drop Maple because they say, "We distributed Matlab and Maple. Well, we don't distribute Maple any longer because Matlab includes Maple, so why are we going to distribute Maple?" In an indirect way, the fact that Matlab was much larger and was including Maple, although that was supposed to be a happy relation, in the end I think ended up harming Maple.

I don't even know if we ever came out with a product that included Matlab for general users. I think that the fast numerics has always been a high priority item in Maple, and it was achieved a couple versions ago with inclusion of the NAG libraries in some form. The MathWorks/Mathlab/Maple relations started on a very good foot. The companies were friendly to each other, were cooperating. Many distributors were encouraged to distribute Maple and Matlab simultaneously. But I think that in the end it may have backfired a little bit for Maple, as I said, by distributors thinking, "We are making our real money with Matlab. If a user wants Maple, we are going to convince them that they have Maple in Matlab and we don't need to carry two products."

HAIGH: I'll ask you later in more detail about each of the other companies you've been involved with. But I was wondering, as this experience with Maple was the first of them, was there

something that you discovered about yourself or what you found satisfying through this experience that led you to be involved with other startups later on?

GONNET: I have started many companies. I have started eleven or twelve companies all together. A couple were collapsed and merged, and one was closed in an orderly fashion. None of my companies ever went bankrupt, so that's something important to say. I don't consider myself entrepreneurial or anything like that. The number twelve may appear a bit inflated. It's a high number, but it has to do with the fact that, for example, Open Text itself had two incarnations. Maple was a company, but for reasons of personal structure we had private corporations that owned the shares of Maple. We had a corporation structure that involved some companies that were holding companies, more than active companies. So it's not twelve companies that I have started that are active companies. Probably six or seven of them are active.

I have all sorts of mixed experiences with companies. I have good experience and bad experiences. I think that the positives exceed the negatives. You're always able to build something that lasts. It's sad to see that sometimes greed and incompetence takes over. Overall I am proud that Maple is still out there. I wish that they would be stronger. I'm actually sad that they have missed opportunities so much. It's not because I have cut the ties, I don't care, or I wish them bad. As a matter of fact, I continue to have quite a good relation with many members of the technical team. We keep our technical relationship, and some of my trips to Canada I go to the company and give talks. Last Thursday, so a week ago, I was there giving a technical talk. So yes, I have cut off my commercial ties with the company. I'm not a shareholder. I have sold all my rights. I don't participate in any management or any advising or anything. I just go there and discuss technical topics with the people. I am happy at doing that. I think that they are also happy doing that. A student of mine, Laurent Bernardin, is now the vice president of R&D of Maple. We have obviously a quite a good relationship.

HAIGH: In general terms, how would you contrast the kinds of personal satisfaction or reward or feeling of accomplishment that you might get from involvement with the companies with the kind of satisfaction or reward that you would get from your academic and teaching work?

GONNET: I think that's a good question. It's different, and it's the same at the same time. It's rewarding. Every time that you build something it's rewarding. I think that that's the real reward. You build something that works that's useful for some people, and that's what is rewarding. When you do some research, you find a theorem, you prove a theorem, or you find some relation that is interesting, and you discover something, you are also building something. When it gets published, that is the confirmation that this is something worthwhile that you have found, that you have built up. I guess in that sense they are the same. The fact that Maple has 120 people is a reason to be proud of. The fact that they are still running the kernel by and large as I wrote it is a reason to be proud of. The fact that there are thousands or tens of thousands of lines of code that are my code that are still being executed is a reason to be proud because it's something that you have built and still exists. From that point of view, they are similar satisfactions. I think that the dominating aspect is that what gives you satisfaction is building something that solves problems for others. Basically building something that has consumers.

HAIGH: Unless you have any other comments to add on Maple, I would suggest that we return to your academic career and discuss your transition to ETH in Switzerland. I believe that you first came here in 1989 as a visiting professor.

GONNET: That's correct. There was a bit of a funny anecdote. With the OED project and with the Maple project we were sort of a favorite target to give demos. The OED, I have to agree, was the best demo giver that we had because it had so much information, and we could search it so efficiently that you would always find something to make someone happy about it. There are hundreds of anecdotes about the OED being successful at demos. So one day I get a request. "Could you give a demo to whoever is coming from Switzerland?" I said, "Sure. Will do," like fifty other times. The day that the people were coming, I discovered that one of the persons that was coming was Klaus Wirth, who you met yesterday. I said, "Oh. Good grief." This is a very important computer scientist that is coming. You told me that this was Hans Buehlman (the president of the ETH at the time) and company, and you didn't tell me the most important part. Of course this was somebody who transcribed the names but didn't know who these people were. I had met Klaus Wirth before, but in much more casual situations.

So we ended up giving them a demo and so on. They were quite interested in what we were doing, and didn't make too much about it. They were busy, they had a tight schedule. We talked for a while, I demoed Maple, demoed the OED. They left. The next thing that I hear is I have an offer to come to Switzerland. They were on a job hunt trip and a candidate hunt. Somebody calls me, and I said, "Look. There is no chance that I can go to Switzerland. I am very happy in Waterloo." Next thing that I hear is Klaus calls me and says, "I think you should reconsider it. It's a nice place to work. I am very happy here, and I think you will also be very happy here. You like to develop software. We have quite a bit of experience here, so why don't you come for a conference?" It was a conference that was celebrating the 25 years of computer science. I came to this conference, and I have to say that all that I saw I liked.

HAIGH: Was that your first visit?

GONNET: Yes, this was just a courtesy visit. I came here for this 25 year celebration, gave a talk, met with people, and so on. Then I got an invitation, well, a standing offer. I said, "I really cannot accept that offer." But the president said, "Why don't you come for a year? You will like it, and you will stay." I said, "Okay. Fine." It was about time to take my sabbatical. It was sort of an early sabbatical, and I took it as a sabbatical year. We came all to Switzerland. I have to say that he was right. I liked it. We stayed.

HAIGH: What was it that convinced you?

GONNET: I have said this many times, and it's summarized in a very simple phrase: In Canada, professors are liabilities; in Switzerland, professors are assets. That phrase summarizes it all. From the government to the institution to your colleagues to your friends, the students, everybody—it's just that. In North America professors are not in high esteem. Professors are viewed as parasites who live in ivory towers that somehow other people have to pay for. In Switzerland, professors are viewed as assets. Professors are the ones who are teaching their children to become professionals, Switzerland depends on good professionals, so they deserve respect, care, and to some extent admiration (remember that ETH has collected 21 Nobel prizes in its history). That makes a huge difference, not because somebody treats me differently, but because the whole system works in a different way. The whole country has more respect for academic values, has more respect, funds better, helps people, and so on, whereas in Canada everything was a fight. I guess that in the States it's also very much the same depending on where you are. In Canada in my last years it was really difficult to get anything done. You had to fight a stupid government that thought that if they would cut universities they would get no penalty from the voters, which was true. They would just cut the universities, and when the

university was just about to recover from the previous cut they would cut again and force this and force that and make things more difficult. It was a constant battle. I don't need that. Nobody needs that.

HAIGH: How would you describe the general position at ETH in the computer science and mathematics level?

GONNET: ETH is a very prestigious place within Europe and within the German-speaking community. We are privileged in some sense by having 150 years of quite prominent history. We have Nobel prizes all over the place. A couple years ago Kurt Wüthrich had a Nobel prize in chemistry.

[Tape 4 of 6, Side B]

GONNET: Earlier on, Richard Ernst had another Nobel prize. Klaus Wirth has a Turing Award. In the German-speaking community (or in Europe) ETH ranks at the top of the computer science departments, so it's very attractive to come to a place that is definitely ranking higher than where I was coming from. It has obvious advantages when you go to recruit students, when you go to recruit professors. People are more likely to come to a place that is one of the top places in Europe. The economy only helps. The standard of living of Switzerland is high. That means salaries are good. That means we have resources that we would not have in other places. From that point of view, the institution has a lot to offer to its professors.

There's also a philosophy that is very different from North America with respect to universities. It has to do with tradition, it has to has to do with the way that it's organized, but it's summarized by the fact that we have only one boss. Our boss is the president of the university—nobody else is my boss. My boss is the president of the university. That's the only person that can call me and give me a hard time. That's the only person that can fire me. That's the only person that can change my salary or do something to me. There are department chairmen and so on, but there are no deans, nobody that can affect my life. So the structure's very simple. The president actually is responsible for all the nominations. Nobody nominates anybody here. It's only the president that hires people. It's a small institution. We are 280 professors. The equation is very simple: We (ETH) are going to protect you from all the bureaucratic garbage, but you have to perform. If you don't perform, well, it's your own problem. We want you to perform. We are going to clean the slate for you. You don't have to worry about anything. But you have to be a good scientist. I love it. I have to say, that type of equation, I love.

HAIGH: You've already mentioned that you brought with you some work on the OED and on Maple and some collaborators. Was there anything about ETH and the department here that led to any shifts in your personal research agenda?

GONNET: Yeah, there was definitely something that ETH helped me change. As I arrived here, I became a very good friend of a colleague in chemistry, Steve Benner. He was doing organic chemistry. In particular he was doing biochemistry, and he immediately saw that I had something to contribute to what is now called bioinformatics before it was even called bioinformatics. As early as 1989, 1990 we were working on problems that we can call bioinformatics. We made an excellent team, and we produced a very significant amount of work. Unfortunately, Steve went back to the States, went to the University of Florida. We kept on cooperating, but it was not the same as when he was next door. So yes we still cooperate, but very little. The big time for our work, and a very pleasant time I should say, was when he was here. I guess that we were both very respectful of each other, of the knowledge that each other had. I am very respectful of his

chemistry knowledge, and we were making a very good team in the sense that he was bringing all the biology and all the chemistry and a great intuition, and I had a background of algorithms and mathematics that could sort of formalize things in some sense. For me, it was really a very good experience, and a very pleasant one.

I have mentioned the importance of having users for what you develop, whether it's a theory, whether it's a theorem, whether it's an algorithm. This was perfect in the case of bioinformatics, because in bioinformatics you don't do things for the theory. Well, some people do things for the theory, but in bioinformatics you write algorithms, or you devise methods or whatever, to solve real problems that the biologists want solved, or that are going to answer real questions in the science. So this is extremely exciting and extremely interesting for me to be able to crack a difficult problem and to see that this solution is going to be helping somebody, or is going to clarify some relationship, or is going to help understanding of how life goes. Bioinformatics is excellent from the point of view of my most desirable mode of operation, which is you want to do something that somebody wants, that somebody's going to use, that somebody's going to consume, whether it's scientific or technical or mathematical. I am quite happy to work in bioinformatics. The major strength right now of my group is in bioinformatics.

HAIGH: Is that the Computational Biochemistry Research Group?

GONNET: That's correct. Steve and I formed the CBRG.

HAIGH: I saw on its website that its main project appeared to be the Darwin project. Is that correct?

GONNET: Right. Actually, Darwin is extremely relevant to the Maple story, because it goes as follows. In bioinformatics you manipulate objects. These are sequence, genomic information. And you have lots of algorithms that are suitable to work with this information, but you still need a language that puts everything together. Very early, in 1990, it was clear to me that if we were going to make significant progress in what is now called bioinformatics, we had to develop a language that would handle these objects as opposed to just write programs in a standard language like C, where every program that we write we have to start from scratch. That was the origin of Darwin. Darwin was really a Maple kernel which was stripped of all the mathematics, and were replaced by bioinformatics functions. So instead of having polynomials, we have sequences. Instead of integrating functions or solving equations we align sequences or construct phylogenetic trees, etc.

The languages, if I show you some code in Darwin and code in Maple, you cannot recognize the difference. The languages are practically identical. The kernels were identical except for where they had to differ up to 1994. I synchronized the kernels for the last time in 1994 because there was a lot being learned and a lot being improved in Darwin and a lot being done in Maple simultaneously. It was a waste to have these kernels independent of each other to rediscover bugs or miss improvements. As a matter of fact, when I was last week talking in Waterloo to the Maple people, I was telling them about some improvements that we did in garbage collection that I am sure are problems that they have, and they confirmed, these are problems they have.

We managed to interbreed, in some sense, the two products to the benefit of the two sides. The products are very different, of course. If on one side they are very similar, on the other end also they are very different. Darwin solves problems with sequences, problems with genomic information; it doesn't do mathematics. Well, it does statistics and it does linear algebra because that is something that is needed massively in bioinformatics, but it doesn't do any integrals, it

doesn't do any solving any equations or anything like that. The domains of application are very different.

The paths of development after 1994 were also very different. Maple was controlled by the company, they had some agendas, they moved the language in one direction. I had other goals in mind. I wanted to make the language more object-oriented. I moved the language in a different direction. Right now the languages are much more different than what they were in 1994, or in 1990 when they were really a single piece of code. Also, around 1994-95, we formalized an agreement with Waterloo Maple in which we would both use the code base that we had. I am not going to do any computer algebra with Darwin. Waterloo Maple is not going to do any bioinformatics with Maple. Anything else in the middle we can all do without interfering with each other. At least we have sort of a formal understanding of what is owned by whom and what we can do and what we cannot do.

HAIGH: Was the Darwin code distributed without charge?

GONNET: The Darwin code is distributed without charge, and it has the same structure as Maple as having a kernel and a library. The library is distributed in source form. The kernel is not distributed. Right now I cannot distribute it without changing the agreement that I have with Maple. As a matter of fact, similarly if Maple wants to make its kernel open source, they'll have to ask for my permission, which I would give them. But we both have a symmetric guarantee and obligation to keep the kernel private because there is so much material that is common in the kernel that if I disclosed the kernel, Maple would definitely have part of its kernel disclosed.

HAIGH: Is Darwin still under development?

GONNET: All of these languages are still under development. I would say that Darwin is more under development than Maple. Maple is more a production system.

There is a substantial difference between the Maple paradigm and the Darwin paradigm. Maple became much more popular much more quickly than Darwin. Why is it? What am I doing wrong that Darwin is not as popular? After 14 years of Maple, Maple was probably used by a million users. After 14 years of Darwin, Darwin is used by a few hundred users at best. Why the difference? There are many reasons for explaining this difference. These throw some light on the Maple project and the Darwin project, so maybe they are worthwhile mentioning. One important difference was that Maple was providing something that nobody else was providing. Let me explain this in the following way. If you have a function and you don't know its integral, there is an integral, and there is only one integral. It's very easy to verify that is either there or not. There is no waffling, right? You either compute the integral or you don't have the integral. If you are not clever enough to find the integral by yourself, or by looking at the table, you are at the mercy of Maple. Maple is only one that can tell you yes, there is an integral; or maybe no, there is no integral for this function, so don't waste your time, there is no possible integral for this function.

In Darwin and in bioinformatics things are much more approximate. You build a phylogenetic tree. Well, the tree is good, the tree is not so good, but that depends on the data. Lots of people can build trees. Some trees are better than others. Yes, I think we can compute the most accurate trees around, but everybody builds trees. It's not like the integral, where it's yes or no, and if it's no you're out of luck. In bioinformatics things are always, "Oh yes. Maybe yes. I have something that is good enough. Yes it will cut it. No it will not cut it." It's a much smoother type of transition between right and wrong.

That's one very important aspect because some people want very precise computations in bioinformatics, but also a lot of people just don't care. Oh, well, I would like a phylogenetic tree, but any good approximation's better than nothing. Whereas, that, in mathematics doesn't happen. If you want an integral, you don't want an integral that may be wrong. This is out of the question: "I'll give you an integral. Oh, yes, it may be wrong. Yeah, but it's not going to be wrong by much. It's just going to have a couple numbers different." "No, you are kidding. This is not acceptable! An integral has to be the right integral!" Well, it could be a numerical approximation, but not a formula that sort of looks like the right thing. This is just not acceptable. In bioinformatics, "Oh, you have an alignment. It's not the perfect alignment. Well, yes, it still does the job." I don't know, nature has played dice with me, so there are some random mutations, so it could be that alignment or it could be something similar, so I don't care too much. Absolute precision is not existent in bioinformatics.

So that's one important difference between the two domains. In one domain you either use MACSYMA, or Maple, or Mathematica, or whatever you have, or you are out of luck with a lot of paper and pencil and a lot of tables, which most people will not be able to do or will not have the time to do it. In the other area you have lots of tools, that may not be perfect, but will still do a job for you.

The second important difference is that the users of Maple are either mathematicians or engineers or computer scientists, and basically all those people know how to write a program and actually have no hesitation of writing a program to do their computations. But biologists are not programmers. They have a lot more difficulties in writing a program. They will actually hate to write a program. They will prefer to point and click, but they will not want to write statements. This is a huge difference that it took me a little while to recognize. A typical Maple user has no hesitation in saying, "Oh, I want to compute this. Then I want to integrate that. Then I want to do this and this and this." Automatically he's writing a little program that's going to do these computations in Maple. A biologist will not be mentally prepared to write a program. They'll say, "No, no, I want something that does my job. Does Darwin do my job?" No. "Well, you have here all the building blocks. You have to build it yourself." "No. I cannot do this." That's a second fundamental difference that I think has affected the acceptability of Darwin in the community.

HAIGH: Are there other packages providing the same basic areas of functionality that have proven to be more widely adopted?

GONNET: There are lots of packages doing lots of different things in bioinformatics. I would say that the common denominator is not a package like Darwin. The typical package is: "I have written a program that does trees in these circumstances. It accepts input in a wide variety of forms. I am distributing it for free. It's up to you to make good use of it." That's one mode. Or else, "I have a web server where I have put all my algorithms. If you click and feed in the information, I will do the job for you." There are many, many web servers (some of them are quite good) that give services to people. As a matter of fact, biology and bioinformatics has used the web much more extensively than computer algebra and mathematics. By the time that mathematicians were awakened to the fact that there was an Internet and they could find useful information, biologists were systematically exchanging data, databases, searching services, computing services, and so on, through the Internet. On one side the biologists will not be able to write a program. On the other side they are far ahead sharing resources through the Internet. Sharing not only data resources, but also computational resources through the Internet.

HAIGH: How large would the potential user community be for Darwin?

GONNET: It's quite large. It's difficult to say. By the same argument that I made earlier, it's an area that is going to be extremely competitive in the sense that a lot of people have written programs that do things, and maybe they are better at doing some particular operation. Is Darwin going to be better in all its capabilities? Probably not. We have quite a few facilities to integrate other people's software into Darwin. I think that the success of Darwin is going to hinge on the "glue" that we have, that is the ability to work with other programs/data/services. We can treat all these objects in a single language, and some of the algorithms that we have implemented in the kernel are basic algorithms developed in the group, and they are good algorithms. But it's a challenge. I don't have any commercial plans for Darwin at this point. I am more interested in developing the system and having other people use the system.

HAIGH: How large is the Computational Biochemistry Research Group at this point?

GONNET: We have eight people here working on payroll: one post-doc, six Ph.D. students, and one staff member. We have a new assistant professor that has joined a month ago. He will bring three more positions, for a total of eleven. We cooperate with a colleague in biochemistry, Prof Yves Barral, and he has his reasonably large group, but not all of his group is involved in the projects that we are working on. Yves is really an associate colleague; he is not part of the CBRG. Then in computer science there are three other colleagues, two in computational science, one in theory, who are interested in bioinformatics problems. We have seminars together and so on.

HAIGH: Did you find that the smallness of ETH made this kind of interdisciplinary research easier?

GONNET: Yes. I think that the smallness, the independence that we have and the very reasonable administration that we have makes things very possible. Having pressure to produce results is very desirable at some point, but when the pressure becomes silly and forces you to do worthless things just to satisfy some administration or some granting agency, then it's counterproductive. I think that from that perspective, ETH has the right combination because we have the pressure to produce, but not the pressure to produce tomorrow ten papers because we need to show them to the granting agency or else we are not going to have money to buy paper tomorrow.

HAIGH: So for your work have you had to seek outside grants?

GONNET: I had, but I didn't need to. I had done a little bit of the rat race, but I really didn't need to. My own regular budget will cover for most of the positions that I have and will cover for the equipment that I need and so on.

HAIGH: You've already talked about the use you've made of the Maple kernel on the Darwin project. I was wondering if you had discovered any other conceptual relationships between your work on algorithms and text processing and this bioinformatics program.

GONNET: No, I don't think so. Now that I have worked several years in bioinformatics, the work in bioinformatics can be summarized as: you have to be good at algorithms, and you have to be very good at probability and statistics. You are not working with completely deterministic objects. You are not working with mathematical formulas that go only one, you are not working with problems which have a unique and precise answer. You are working with nature that has gone into a process of evolution in a relatively random way. This randomness percolates

everything that you do because this randomness is not only in nature, but in all the data that you acquire. You acquire data, and the data is not exact. It's subject to error because of the nature of the data or the nature of the acquisition of the data. What I tell all my students and my grad students when they come is to make sure that their background in algorithms and their background in probability and statistics are really strong. If they have a good background in algorithms and statistics, quite a bit of scientific computation helps. It helps if someone knows how to integrate a system of differential equations or finding a minimum in an efficient way. Those kinds of basic scientific computation abilities are also very helpful. But if you are good at those two and possibly that third one, you are going to be good a bioinformatician. There is no two ways about it. But you have to understand algorithms and statistics, and that's maybe the crucial point.

HAIGH: You've described the work of the Computational Biochemistry Research Group and in particular the Darwin project. Have there been any other aspects of your academic career at ETH that you would like to discuss?

GONNET: No, not in particular about ETH. But talking about Maple and talking about Darwin it comes to my mind that at a certain point there was another descendant of Maple, which because the OED project stopped its activities, and because the company Open Text was not interested in it, never saw the light of day. We had the same approach for text searching and intelligent text processing we took to Maple and for Darwin, and in that case, we designed a language called Goedel. This was a bit of a pun because it had the OED in it and Goedel is a famous mathematician, very important to computer scientists. The language Goedel was, as Darwin is, a descendant of Maple. It had the same language structure, and had different primitives to handle text. Some important projects for Oxford University Press were coded in Goedel and were executed in Goedel. Unfortunately, this language died, from nobody continuing to work on it. The normal place for the work would have been the company Open Text, and Open Text did not have the energy or the interest to follow this approach. So when we talk about Darwin as a descendant of Maple, we should at least mention Goedel, which existed for a short period of time.

Goedel was used for a significant project for Oxford University Press. There is one dictionary which is a derivative of the Oxford English Dictionary which is called The Shorter OED, which as opposed to the other dictionaries is not done from scratch. Most of the Oxford dictionaries are done basically from scratch. The Shorter is in reality a shorter version of the OED. They painfully wrote down all the rules that were used to derive an entry from the big OED into one entry of The Shorter OED. This was quite a complicated procedure that was implemented in Goedel. This created the frameworks for the entries in The Shorter. Oxford University Press was very happy to have this done automatically. It was probably the only big, successful application of Goedel.

HAIGH: You had also signaled out your paper on the Lambert W function as one that you would like to talk about. [Robert M. Corless, G. H. Gonnet, D. E. G. Hare, D. J. Jeffrey, and D. E. Knuth, "On the Lambert W Function," *Advances in Computational Mathematics* 5, 1996, 329-359].

GONNET: The Lambert W function is somehow very close to Maple and very close to my research and a nice event all together. Let me explain, first of all, that the W function is the solution of a transcendental equation which happens to be very handy to solve a wide class of problems involving asymptotic expansion, differential equations, and a few other mathematical

problems. The W function had been studied by many mathematicians through history, Lambert being one of them, Euler being another one, more recently an Englishman by the name Sir Edward Maitland Wright. So none of us discovered the W function. The W function existed for centuries.

My contribution was to create a framework to include it in Maple from the very early stages. I included it out of necessity because it was appearing in many of my problems in analysis of algorithms. It was actually very handy to have this function as opposed to having approximations of this function, which is what most other people were doing. At that point I realized something which is sort of obvious: that many of these transcendental functions are just really names with a certain number of properties attached to them. What is the difference between the logarithm function and the W function? Well, formally none except that the logarithm function is very well known—even high school children will know about the logarithm function, and the W function is a bit more obscure. Both have mathematical properties, both have inverses, both have series expansions, both solve differential equations, and so on. They are indistinguishable from a mathematical point of view. Once you give them a name, they come into existence. You can compute numerical values, you can compute special values, and so on. Everything that works for the logarithm works for the W; everything that works for the W function works for the logarithm. It is in some sense that very simple realization that you just need to name a function that allowed you to put in the mechanisms so that is now possible to treat many objects that were not possible to treat before as if they were as common as the logarithm function. From the point of view of computer algebra, the W function is important in itself, but I think it was more important in making us realize how to deal with unknown functions: that once you know certain properties and you give them a name they are as common as your normal functions: sine, cosine, tangent, logarithm, exponential, and so on. They become just one more of the group.

So the naming “W function” comes from a paper in communications at the ACM, and the name of the W function was a source of some debate. In the end, this paper that I took as a sample used an italic W (or maybe an omega, but I think it was an italic W) for describing this function, for which everybody who described it was using a different name. Once you incorporate a function in a system and people start using it, it starts taking a life of itself. So the W function started taking a life of its own. At some point Maple decided to call it “LambertW” because Maple did not like one-letter names because one-letter names are reserved for users that want to use them as variables, but for all the people involved closer to it, it’s the W function.

At some point while meeting with Don Knuth we had a discussion about this. Knuth uses another version of the W function which is formally extremely similar to the W, but he calls it T. For him it’s a tree generating function. We had a friendly discussion of whether I was going to convince him to use W or whether he was going to convince me to use T. In the end we decided that there was a small difference and that we will both establish the relation between the two functions and actually publish a paper together, and that’s what we did.

It turns out that the W function appears in many, many places, in engineering, in science and engineering, and physics. This little piece of work became quite popular, and the W function is one of my most cited papers currently. In some sense I am very happy to have brought the W function to life in Maple. Of course I didn’t invent it, I only implemented it and discovered some of its minor properties, but definitely implemented it in Maple. Also this sort of opening of the door so that any function that is suitable, that will explain some mathematical property, can be given a name as soon as you know enough about it. Knowing enough about it summarizes to

know its mathematical properties, to know its integral, its derivatives, its series, and how to compute it numerically. Nothing more. Once you know those things, you can give it a name, and it's a function that will exist by itself.

HAIGH: I also remember that listed on your resume are a number of books on Maple. Did you put a significant amount of time into working on those?

GONNET: I put some time. I would say in all fairness, Keith in particular, but all the other coauthors put in a larger amount of work. I had contributed or I had devoted more effort to the system itself than to the books. Yes, I have been an author and I have written some minor parts of the books, but the main effort in writing the books was by my coauthors, not myself.

HAIGH: Is there anything else you'd like to say about your work at ETH, or perhaps your graduate students here or colleagues?

GONNET: No. I think that most of the things that I wanted to say have been covered already.

HAIGH: Let's move onto address the OpenMath Project then. Perhaps you could begin by talking about the initial impetus for the project and what time period it evolved over.

GONNET: The OpenMath Project probably can be traced to the mid '80s when we realized that we needed a language to communicate mathematics between different systems. Whether the systems were computer algebra systems or numerical systems or storage and display systems or simply storage systems, we needed a way of communicating a formula, a matrix of data, or something from a system A to a system B. That was recognized as a problem. There was no standard for doing that. There was no standard for displaying this information. There was no standard for plotting this information if it was a function, and so on. We wanted to establish a standard which was a little bit more general than just Maple, because Maple would just be viewed as a proprietary format, and Maple was not covering all the possible aspects of data storage and transmission. We perceived that there was a huge need from the community to have this Esperanto for mathematics, this common language in which you can transmit your formulas. How can you get a formula transmitted from Maple to Matlab, and then from Matlab to a paper into which you want to incorporate it? That was not available. Individual tools to output Maple into LaTex suitable for a paper were available, but a general language for describing mathematical objects was not available. This idea was already present in the Symbolic Computation Group I would say in the mid 1980s, but nothing much happened at that time.

HAIGH: Had you been influenced by your personal exposure to SGML with the OED project?

GONNET: Yes. Absolutely yes. There is no question that treating data as a structured media was definitely influencing me, and that was basically what the OED project had taught us to do.

[Tape 5 of 6, side A]

GONNET: This was not a project that anybody would actively pursue in Waterloo. Even though we were all in agreement that this was a good cause, nobody was able to do anything with it. It was only when I was in Switzerland that at some point in time I invited several colleagues, and at that point we coined the term "OpenMath" for this language that would allow us the interchange of mathematical objects between systems. This was quite a successful project at first, and people were very enthusiastic. We started getting the right ideas in place about how to represent and how to solve the difficult problems. The difficult problem is the semantic problem, how to represent different mathematical objects and being able to transmit mathematical objects from one system to another. Accepting the fact that different systems may have different capabilities,

while Maple can compute an integral, Maple probably will have problems displaying some function, and Matlab will have problems integrating symbolically an expression whereas it's not going to have any problems in computing the eigenvalue decomposition of a matrix and so on and so forth. This work went on, and very quickly we created an OpenMath society and created some formal structure. We were meeting every year. The first two meetings were here in Zurich. It was mostly an effort that had more members from Europe than from North America.

HAIGH: Were the participants primarily producers of mathematical software or users?

GONNET: It was mostly computer algebra related people more than anything else. Cleve Moler, for example, at some point was very interested, but he was sort of waiting on the sides to see what we produced.

HAIGH: Did you see yourself participating as a representative of Maple, or just as an interested citizen with expertise in the area?

GONNET: I saw myself as coordinator of that group. At the time, I was basically providing most of the energy for this to happen. At some point we had some funding and we were able to hire some people part-time in different institutions in a distributed way. But it has to be said that unfortunately nobody took the flag of OpenMath forward with all their energy. Nobody made of OpenMath their top priority at the time. With time, I got a little bit frustrated with the following situation. We would meet one year. We would discuss. We would make progress. We would define some properties of the language that we had. Then nothing would happen for 11 months and a half, and then we would meet again, and we would start again from zero. All of the discussions, all of what we have done, all of the conclusions that we have reached, were all forgotten and we were all starting again from scratch. Then repeat. We would make progress, we would define things, come to conclusions, and so on. Leave the meeting, 11 months and a half will pass, nothing would happen, and then again we would meet and we would start from zero again. At some point I got tired of that, and I gave up on the OpenMath project. My view is a little bit pessimistic. It's not that we didn't achieve anything, but in my view we were not making progress at the speed that we should have made progress.

At the time that I left, there was also sort of a competing project, which was the representation of mathematical objects in HTML, which was called MathML. This had a different goal than OpenMath. OpenMath wanted to be able to represent expressions and what they mean, as opposed to MathML, which wanted to represent expressions and how they are going to be displayed on a screen. Which for some people may appear to be the same thing, but in reality they are different. We were interested in being able to do computations with the objects and to do the right decisions. We didn't care how this object was represented on the screen. Whereas Math ML people were concerned about how to represent objects on the screen, they were not concerned at all about what they meant. So for them, they were completely happy of having some display in two dimensions that made mathematically no sense whatsoever, if that was what the user wanted to display. We would say "No, I want to understand whether that's an integral, whether that's a differential equation, whether that's a matrix. I want to understand what it is and I want to be able to tell some other system what this object is."

So the two standards coexisted for a while, and actually one would be included in the other. If you wanted to understand what the meaning of the object is, MathML allows the inclusion of OpenMath in it. OpenMath allows the inclusion of MathML if you want to describe how you will represent this object. As I said, at some point I gave up on OpenMath.

HAIGH: What year would that have been?

GONNET: That would be about five years ago or so. I gave up on OpenMath for the reason that I mentioned earlier, that every time we would appear to make progress, and then because there was no person carrying the flag, everybody would go back to their own projects and everything would be forgotten. I was frustrated with the lack of progress, or the lack of progress at the speed that I thought should happen. And at some point in time a European grant was applied for, and somehow they left Switzerland outside and I used that opportunity to say, “Okay. Fine. You’re on your way. You’re on your own,” and left it. I thought that nothing was going to happen. To my surprise, OpenMath is alive and well these days. I don't know if it’s thriving, but it’s definitely alive and well. Maybe when I gave up on them… I gave up on them not because it was a bad idea, or it was the wrong thing, but because I couldn’t see how to move it forward, and I was also sufficiently busy as to be unable to say, “I’m going to carry the flag of OpenMath.” I had enough projects for myself to be busy with. Although I was mostly the convener and the starter of all the OpenMath projects here in Europe, I just couldn’t carry this project forward.

HAIGH: Are you aware of whether there was anyone in particular who stepped forward and took up the burden?

GONNET: Yes. I think that I have to make special mention of Arjeh Cohen and Mika Seppala. They were very active in promoting OpenMath, and they basically carried the work forward. James Davenport in the UK at Bath also played a very important role. There were lots of other names, but those three names come to mind as people that continued moving the project forward. And as far I know, right now they are very active. Last year I was seriously planning to attend their meeting because I think that last year they were celebrating ten years of OpenMath. I was very sad that at the last minute there was a conflict and I couldn’t attend the meeting, because I really wanted to see now how this child that I had abandoned continued its life and prospered in the end.

HAIGH: Are you aware of whether support for OpenMath has now appeared in the main commercial products?

GONNET: No. I know that Maple supported OpenMath, but in a very quiet way. At the time it was also one of my disappointments that I could not even convince Maple that this was a worthy cause. Many people would pay lip service to it, say, “Oh, yes it’s great,” and then proceed to do nothing. In some cases even showing signs of thinking “Well, you know, we don’t want our machine to be used for the algebra and then use some other engine for displaying the math.” Some people were worried that accepting and interchanging a common language would detract from Maple. That was a serious mistake in some sense. I think that systems would have been much better if they were very good at something and people were using them, as opposed to try to do everything for everybody and being mediocre in most things. In the end, the computer algebra systems are completely monolithic. Every computer algebra system wants to do everything, and they don’t talk to anybody else. This is what Maple is doing and what Mathematica is doing and what everybody else wants to do. They want to solve all the problems for you. They want you to buy only one package. They will not communicate with a numerical system or with a general display tool. No. They want to do everything themselves. At the end there is a waste of resources, a tremendous waste of resources.

HAIGH: The main topic remaining to us would appear to be your involvement with the other firms that you founded. Chronologically I would imagine it would make most sense to talk about the Open Text Corporation next.

GONNET: Open Text was a really very peculiar experience. It has similarities with Maple, in that this was a university project. In this case it was the Center for the New OED project. It was centered around some software that we had produced, searching software and some other tools that were also interesting. It had a somewhat similar start to the Maple company. We incorporated a company where most of the shareholders were professors or members of the research group who put up some very minimal amount of money to start the company.

With Open Text, very quickly there were some differences. The type of customer of Open Text had a completely different profile from the customer of Maple. The customer of Maple was a researcher that was happy to throw $500 or $1,000 to a software company, and would usually be sort of a scientific or an academic user of the software. Text searching software was difficult to sell. It would sell to businesses, and it would sell for huge amounts of money, compared to the $500 or $1,000 that we would get for Maple. It was a completely different type of sale. It was a completely different type of company profile that you had to have. I realized that the company was not going anywhere because my colleagues were a little bit too conservative. I remember having a discussion. “Should we port to the PC? I think we should port to the PC, I said”, and my colleagues said, “No. That’s going to cost money.” Well, “I will invest the money, I said”. “No, no. We don’t want more investment (that will unbalance power). We want to go slowly.” Well, the slower you go, the more opportunities you lose.

HAIGH: What platform was this operating on?

GONNET: It was a typical UNIX platform. The short of that lack of initiative was at some point I got tired of my colleagues being so conservative, and I decided to start a new company. I found a person who was a real business manager, who really impressed me by the way that conducted himself and that he presented himself and so on. I was already living in Switzerland at the time. This person flew to Switzerland to show me how responsive he could be. I remembered Tim Bray calling, “I have interviewed this candidate, and he’s good, his name is John Branch, guess what…” Sure enough, when I read the message he was ringing me on the phone. He was here in Zurich to see me. I think Tim said something like, “You’re going to have a surprise,” or something like that. We started a new company called some opaque name like International Retrieval Systems Corporation.

HAIGH: Was Open Text the name of the first company?

GONNET: The name of the first company was Open Text Systems. When I started this new company, all my colleagues saw the light, and they said, “Well, it doesn’t make sense to have two companies.” Or actually they were probably feeling that the old Open Text would go bankrupt very soon because I was going to put all my new efforts into this new company and I was going to call the shots. We arranged for a merger of the two companies on the conditions that I would be in control in the new company, and the share structure would reflect that by the time I had invested more money into the company. So I was the only or largest shareholder of IRSC. At that point we founded a third, new, company called Open Text Corporation, which was the result of the merging of the two old ones, the Open Text Systems and the short-lived company that I started. Open Text Corporation is the current Open Text Corporation that is traded in NASDAQ right now.

The company had a reasonably good start and started focusing on text searching software. We had to convince people that text was a valuable asset. Sounds ridiculous now, but people were thinking, "Oh, we throw out our mail messages. Why do you want to keep mail messages around? They use megabytes of storage!" We had to teach people the resource that they could have in their data. For that the Oxford English Dictionary became very handy because we would search almost for anything in the OED and you would find useful information. It was a fun project, and it grew quite well. We had a huge number of opportunities, that unfortunately not all of them came to fruit. When I say a huge number of opportunities, these were real opportunities in nowadays scenarios. We were doing text searching in 1990 way before all these other companies—way before Google of course, but way before even AltaVista were in the market. We were the first company to index the web, full page indexing, full text indexing of the web. We were chosen to be a companion of Yahoo! at the time to do the searching of the content of the pages and be the "index" of the web, and Yahoo! was viewed as the table of contents of the web. The two companies were partners at the time.

Unfortunately, venture capitalists that controlled the company at that time had a different view of searching the web. They made perhaps the stupidest statement that you can make. They (Richard Black) said, "We don't see any viable commercial way of exploiting the searching of the web." Well, we didn't see it either, but we saw that there was something there. We saw that the same thing that was happening when we searched the dictionary and found any information that was useful, we could search the Internet and find information that was useful. We couldn't put it in a piece of paper to convince a venture capitalist or to convince an investor, but we were all convinced that this was a great idea. Unfortunately, with venture capitalists thinking that way, they took resources out of the web searching engine that we had. Eventually the web searcher was so overloaded that it was painful to use. Eventually Yahoo! dropped Open Text, and AltaVista came with the same system that we were designing at the very beginning. The layout of AltaVista was almost a carbon copy of ours. AltaVista became a company that was worth billions, and then also fell in disgrace and Google now is worth a hundred billions in the same space.

HAIGH: Did that search engine ever have its own brand or identity, or was it just…?

GONNET: Yeah, it was called Open Text. It was the Open Text product. People were calling it Open Text. Internally we call it PAT. Our internal term for the searcher was PAT.

The other thing that the project also discovered, and we missed completely was the browser. When we were browsing the OED, the Oxford English Dictionary, we developed a browser, which was working with a searcher. We had a clever way of tagging things so that they would show with the right fonts and the right spacing and the right colors and the right sizes on the screen.

HAIGH: So the browser was a desktop client application?

GONNET: Right. The browser was called Lector. That was the name that we had for it.

HAIGH: Would that software have appeared in the CD-ROM versions of the OED, or did they develop something different?

GONNET: No. In the end, Oxford University Press took a different route for producing a version that was searchable and distributable to people. We were distributing a version of the OED for academic users with our software, but it was not on a CD. It was on big tapes because it was the complete version; it would not fit on a CD. It was also very high priced. We sold several copies,

but it was not a commercial success. In view of that, Oxford University Press contracted some other company to produce a CD with the OED in it in a searchable form, but a little bit more primitive than ours. That was not a commercial success either. The second edition of the OED, the paper version, was a huge commercial success. People loved to buy this book, and paid fortunes for it. So Oxford University Press got an unexpected success from the second edition, but no success from either of the electronic versions. I still have the original OED. I can show you Lector in action. We should do that because I think that you will enjoy seeing it.

Because the point that I wanted to make is the following: We all recognized that there was a huge advantage in having a browser. You search for a document. You find a document. You want to go up and down and highlight or see something. You love to see it in different colors and different fonts and so on. In that particular case, we wanted to show documents/entries as close to the original dictionary as possible.

But we didn't make the mental leap that the browser was the most useful tool that we were developing. Millions or billions of dollars were then spent in designing browsers when we had years of advantage over everybody else. Actually, to some extent, better ideas than everybody else on browsers. That's another point that we missed. In that case we missed because we developed a browser, we realized that it was great to have a browser, and we didn't make the leap that "oh, this may be good for everybody". We should have had a browser that reads HTML. Well, HTML barely existed in that timeframe.

HAIGH: What platform did it run on?

GONNET: At this stage it was mostly UNIX workstations, so it would be typically Sun workstations. I think that two or three small Sun workstations were all the hardware that was assigned to index all the content of the pages of all of the web.

I was chairman of Open Text until about 1994 or something like that. Open Text, in contrast to Maple, was dominated by venture capitalists.

HAIGH: Had they been present at the very early days of Open Text?

GONNET: No, actually they came halfway when we needed some intermediate funding.

HAIGH: Had you been the main initial source of funds for the company?

GONNET: Yes, but still in much lesser quantities. I was not independently rich or anything like that. I was at a point that I could start the company. I did start this IRSC, and then I did start Open Text Corporation, but I did not have the power of funding the company for several years. Pocket money in terms of venture capitalists is what I provided. Then the Ontario government and Helix International became shareholders. There is a lesson that I think that I knew that I ignored. Venture capitalists are very nice people when there is not much at stake. When there is real money at stake, they are ruthless. When they saw that Open Text (this was way before the bubble, but when Internet companies were starting to appear to be very important) was starting to grow they became completely ruthless with everything. The VCs did too many dirty tricks to the original shareholders of Open Text to describe here. They found an ally with the Ontario Government (who then became well paid employee of Helix), used scare tactics to change the Board to their favor, 2/3's of the company shares to themselves at firesale prices, violated the legal rights of the shareholders, etc. They decided to control the company in their own ways. We also had problems with the management at the time. In any case, I didn't want to be involved. I

was removed from the board, and we ended up not in good terms with them (had to take the company to court for violation of shareholders rights).

HAIGH: That was in 1994?

GONNET: 1994-95 yes. The experience with VCs was really bad for me. As I said, VCs are going to smile and do nothing nasty to you as long as there's nothing really at stake. When there is real money at stake they are going to be as nasty as they need to be to take the biggest bite. This is a general rule.

HAIGH: Had the person who came to see you in Switzerland finished up running the merged company?

GONNET: Yes. This was John Branch. He ran Open Text for the first few years. Actually his demise was part of the end of my involvement with Open Text. But Open Text at the time, up to demise of John Branch was a very much fun company to work with. The product was exciting. The things that we were doing were extremely exciting. Open Text has a market capital of one billion these days. If we had been given a bit more of encouragement instead of being criticized on indexing the web…. well, it could have disappeared some people would claim, but it could have also had a market capital of $50 billion. There is no reason why Open Text, which was a partner of Yahoo!, would have not had the same trajectory as Yahoo! if it had been properly managed. That's what is also criticizable. VCs at the time said, "There is no model for making money out of the Internet," so they concentrated on enterprise business, and they moved all of the technology to intranets. So in your enterprise, you want to have management of documents. They relabeled themselves "a document company" at some point, which is something that I was definitely not interested in. As a matter of fact, at some point they even dropped text searching completely.

HAIGH: You discussed this early search engine that was created internally. Through this period of 1991 to '95 when you were involved with the company, did it have any other products or services that had reached market?

GONNET: No. The only product that was reaching the market was the search engine and associated programs, that were, for example, the browser that was also distributed as part of the searcher and so on. It was a single product company. Like Maple in some sense, it was a single product company. There was software associated with making the indices software associated with preprocessing the data and so on, but there was no other main product.

HAIGH: Had your own idea been exclusively for searching the Internet, or did you expect to sell it to businesses as well?

GONNET: No. Our model was to sell to businesses. Searching the Internet was an idea of Tim Bray. He thought at some point, and I remember him proposing this to a board meeting, saying, "We are going to index all the pages on the Internet." It sounded a little bit crazy, but we immediately approved it and thought it was a great idea. He was allowed to start it, and it was done for some time. It was very successful, but then it was killed. It was killed by starving it, because if you don't put the right hardware, if you don't put the people to support it, it dies. I think Open Text at some point was receiving 300,000 requests per day, which at the time was a monstrous number. People were waiting for ten minutes to get an answer. Forget it; with those responses it's not a service. So that was a great loss, a great loss of opportunity.

HAIGH: How large was the company at that point in terms of employees?

GONNET: Twenty to thirty employees. Closer to twenty probably than to thirty.

HAIGH: Had it maintained any ongoing links with Waterloo?

GONNET: Open Text? It's in Waterloo. It's located in Waterloo.

HAIGH: No. I mean with the University.

GONNET: No. Actually, Open Text had fewer contacts with UoW than Maple did. Maple maintained contacts with the university for a lot longer, supporting research groups and so on. I guess that the position of John Branch with respect to the University was the University got a fair deal with the software. I don't remember the details of the deal. "The university got all its glory. Now it's time for us to make a business. We don't owe anything to the University". That was John Branch's position. Also, the Center for the New OED, being an academic project, had a beginning and an end. Once it had reached the end, there was not much more to do. Whereas the Symbolic Computation Group kept on doing research in computer algebra and kept improving the programs, the Center for the New OED basically closed or disappeared or became inactive, so there was nothing really to support at the University. The company was not going to support it.

[Beginning of Session 4, held on the morning of Friday March 18, 2005]

HAIGH: Before we continue with your involvement with the various companies that you've founded and helped to run over the years, I understand that you have some additional points you would add to material that we had covered previously.

GONNET: They are unrelated points, but I think that are worthwhile mentioning or emphasizing if they were not given the proper stress before. I mentioned the test suite that was testing Maple for consistency and for portability. The test suite is not something new that we have discovered or anything like that, but there were a couple attributes of this test suite that made it very significant for us. It's not a well-known sort of concept. These points are the following:

The test suite normally contains what a programmer thinks is going to test his/her software. Usually this is not enough. The programmer has a pattern of what he knows that his or her code will execute and normally produces tests that always pass. What we were doing is every time that we were finding a problem, a bug, a defect, when fixed, this bug or defect was incorporated in the test suite. This caused the test suite to grow significantly. It turned out that the bugs typically reflect a path of thinking that was not the programmer's path of thinking, and it turns out that in practice the old bugs are excellent witnesses for any problems that develop in the future. That was a small but I think not very widely known practice. Regression testing is normally done that way.

The second aspect that was important of test suite is that a test suite can automatically decide whether the results are the same as before or are different. What we are doing is we record the correct results and automatically compare the new output with the old output to determine whether it's passing or not passing. Sometimes it cannot be done completely automatically; it has to be coerced a little bit into a form that compares as equal. What is very important is that there is no human needed to inspect the results and make a decision. That a decision made by a human takes time, and typically the human starts letting some minor differences go by. It's very difficult to judge in the end whether the test is passing or not, or whether this test has slowly wandered away from the desired results.

A third aspect of the test suite, which made a significant difference in various aspects, was to start testing absolutely everything that we could test in ASCII. For example, we were graphing output. The output of a graph is very difficult to test. It's mostly something that a human will look at. It was a mistake not to test it. We started printing ASCII versions of the plots, which are extremely coarse, but now can be compared and can be determined whether we are doing a better job or a worse job than before, or they can be printed in PostScript and the PostScript can be compared to previous results. We know whether there is a difference.

Finally, the final step of the test suite is probably a double step. At some point we designed a person to be in charge of all the problems who was named the "entomologist" because he/she was taking care of all the bugs. The entomologist was responsible for running the test suite every night. Even though it was very extensive and it required many, many hours of computation, it was run every night. That made a significant change in the quality and in the easiness of maintaining the software because it meant that if you had made a change that was damaging the system in some unpredictable part, the next day you would have a message you had done something that broke the system. As opposed to you make some change, six months later somebody says, "Something is broken" and nobody remembers what is the cause. It's very difficult now to go back, assign or find who is the culprit and find the fix. If I fix something today and tomorrow first thing in the morning I have a message that said, "You broke something," most likely I have a very good chance of fixing it very easily.

So these very little things made the software more reliable, and our job at guaranteeing that even within the university and later in the company we had a quality product to a certain extent, made it much easier. The test suite in Maple was very extensive. It consisted of small sessions recorded in ASCII, and we had thousands of such sessions.

HAIGH: At what point did you adopt this practice? Were these in the very early days?

GONNET: The very early days. Since the first days we've had a test suite. The test suite at some point became a very large burden on me because once that the test suite doesn't pass cleanly, you get into a vicious circle that people will run the test suite and they try to fix their problems. Well, not all the problems are theirs. But you get into a vicious circle where the test suite doesn't pass with more and more failures. Nobody's responsible for some of the failures; nobody wants to fix someone else's problems. So you get into the situation that I was describing before: six months later you want to release a new version and you have hundreds of tests not passing and its difficult to find the culprits, it's difficult to fix the software. It's a situation where when you are on top and the test suite is passing and you run the test suite very frequently, you are in top shape. When you don't, it tends to deteriorate, and sometimes it's very difficult to get back to above water.

[Tape 5 of 6, side B]

HAIGH: On the topic of the test suite, were there any other earlier projects that had been particularly influential on your early thinking about this, or were these techniques that you came up with much independently?

GONNET: No. I came up with these techniques largely independently. Test suites had been invented for a long time. Regression testing is a standard word in most software shops. I think the combination of all these little ideas that I describe was certainly unique at the time or not widely known at the time. The novelty was having a large test suite where you incorporate your defects back into the test suite, and where you make efforts to make an automatic decision

whether the test is right or wrong, so that a program can decide and run overnight and quietly verify all the system. And that you test as much as you can of the system, even the user interfaces, by coarsely translating it to text. It is something that I have applied in all of the companies that I have been involved, and I would say to great success. We have a test suite in Darwin. Open Text had a test suite. Aruna and Web Peals also have test suites for their software. Nothing here is completely novel or a great idea. It's just four or five little things that do make a difference.

So I was going to jump to a slightly unrelated topic. This has to do not necessarily with Maple alone, but with the whole area of computer algebra. This has been mentioned many times. It's the relation between computer algebra, in particular in the design of systems of computer algebra and the object-oriented community. I think that design of systems in computer algebra never got the credit that it deserves in the object-oriented community. A lot of concepts of object oriented programming have been commonplace in computer algebra systems. A lot of object-orientation concepts have been available for decades. To all of a sudden have the object-oriented community "discover" these concepts again and make a big fanfare about how novel they are and so on, makes us say "Hey you guys, this was available in MACSYMA—it was available 30 years ago. What are you talking about?" The object-oriented community is very self-centered and doesn't want to hear that something has been discovered before. I have sat many times and listened to someone presenting some topic on the design of modern programming languages, and I say, "Has this guy ever heard about symbolic computation? Never heard that we were doing this 15 years ago?" It's a little bit annoying, and I wish that these people that discovered new concepts would spend a bit of time going back and finding what other people have done.

It's clear that sometimes the terminology's different; it's clear that the concepts sometimes are slightly different. For example, not so long ago, they were saying that object orientation is not the whole thing. We have aspect-oriented programming and so on. Well, this is a discussion that we had in the early days of Maple when we were discussing whether operations should be dispatched by the data type or should be dispatched by the operation? We clearly identified some groups of objects where it was much better to dispatch by the object and some group of functions that was much better to dispatch by the function. For example, if you're working with matrices it's more natural to dispatch by the object. You have operations and matrices that have certain patterns. But if you are differentiating, it's much more natural to dispatch by the operation because the derivative of classes of functions share lots of similarities, so it's much more natural to dispatch by operation in that case.

Well, I'm talking about something that we were discussing in the early '80s. Now the object-oriented community gave it a label and so on, and it's being touted as a new paradigm. I don't know. It's just a bit of a gripe that I have about design of systems of computer algebra not getting the recognition that they deserve in terms of the object-oriented community. I should say that the models that Scratchpad and later Axiom had about abstraction are, in my view, superior to all the models that have appeared of object-orientation. Granted, Axiom, as I said before, was a very difficult language for the users in mathematics and probably outside mathematics because of being so formal and so pure and so exact in its concept, which some of them are not trivial. End of gripe.

A third completely unrelated point that I wanted to make has to do with published and unpublished results. This is a mixture of several things. The Maple project, the New OED project and later the Open Text project and my later project in Aruna with relational databases are filled

with unpublished results. I think that this is a fact of life for this type of project. The unpublished results arise from several things. First of all, you are worried about doing something in the system and have to resolve a problem that is satellite to your main goal. You search in the literature. You don't find something satisfactory. You have to build something or you have to build a new algorithm or you have to improve something or whatever. But your goal is not to publish something in that particular area. Your goal is to get the system to work or to get the system to do something else. Most likely you don't have the energy to later go and do the literature search and write a paper and so on. The goal was a slightly different goal. One example that comes to mind is the garbage collection in Maple. The garbage collection in Maple is quite unique. It's most likely interesting, and it most likely would be a very interesting paper to be published. Nobody ever had the energy. I never had the energy to publish this.

The second aspect is that quite often when companies that are distributing the software, the companies don't want to release information that will make it easier for other people to understand, or copy, or take the advantages. In some cases a few of the algorithms are crucial to understand why this software has an advantage over other pieces of software. You don't want to tip your cards and show how your internals are working and create unwanted competition. This was clearly the case in Open Text. It was clearly the case in Aruna. In the case of Aruna, we wrote three patents on the software, and two of the patents were clearly obscured. The company wanted the protection, but didn't want to tip the cards very much. All the descriptions that the technical people wrote were obscured at some point to make it really difficult for someone who was going to use those ideas illegally just by reading the patent to engineer back the software.

So for those two reasons and maybe others, that is the people that are working have a goal that is slightly different from publishing those particular results. And because the companies become a little bit jealous or a little bit protective about their information, these projects, in my view, are filled with small, medium and large unpublished results. The case that to some extent got me I wouldn't say in trouble, but got some friction in the community was one of the structures that was developed for fast searching in Open Text, which internally is called S-Vectors, which then were rediscovered by several people, but it was rediscovered ten years later. Well, it was a little bit annoying to see that that was happening. What can we do? We didn't publish it at the time or it was very hidden at the time. We don't get the recognition. We did get a company going, but it's a fact. It's a fact of the environment where I have been working that we live with a lot of unpublished results.

HAIGH: Did these all occur at the stages after these software technologies had been passed on for commercial development, or did you find that this was affecting the way you were working even inside the university?

GONNET: No, even inside the university. Let's take the garbage collection example. The garbage collection is sort of a novel algorithm that Maple has. It was written out of need, completely out of need. We had written a garbage collection based on reference counts, and at the time (I'm talking 1984) we didn't have programming support for doing this automatically. It had to be done by hand, and it was impossible to get it tight. There were tens of thousands of lines of code, and there were oversights of marking storage all over the place. It was just an impossible task with the tools that we had at the time. We were about to release to Watcom, and garbage collection still had lots of bugs and it was not working, and there was no light at the end of the tunnel. At one point I said, "No. We cannot distribute with this garbage collection. We have to have a completely different garbage collection system." So I asked everybody to hold on

for a week or something like that. I'm going to write a new garbage collector. It took me longer than a week, but we got a garbage collector that was working, and it was a novel idea. Yes we had to fix a few little problems, but there was never an interest in doing research in garbage collection. Garbage collection had never been my area of research. I never wrote a paper on such. Actually, an employee of the company wrote a paper on that garbage collection so that the people in the company could understand how garbage collection was working. That algorithm is still, as far as I understand, unpublished unless somebody else in the meantime rediscovered it. It's not necessarily the companies; it's just that all of a sudden you need something that you don't have, and these things happen typically out of necessity and typically out of pressure situations.

Finally, the other thing that I said before, I think I want to stress, if we had made a little contribution here I think it's probably the most important one. This is thinking about the scientific computation community at large. It's the fact of recording the mathematical knowledge in a form that is accessible to everybody. All the knowledge about integrating functions, for example, is encapsulated. Let's take Maple as an example. In the functions in Maple you can type "integral" of any function and you basically get all these theorems working for you. I think that this is a real contribution for people that are going to be using mathematics. Instead of having to go to the library and do an extensive search on how to integrate this particular set of functions, and probably a very difficult search in the literature or consulting your colleagues or calling your friends that may know about this and so on, you have that algorithmically available to you. I think that that's the big contribution of computer algebra—making the knowledge of lots of mathematicians that worked for a long time but have put all their results in journals that are very difficult to extract and to find, putting it at the fingertips of everybody. I think that that's one of the most important contributions.

HAIGH: To what extent would Maple automatically select the most appropriate method, and to what extent would the user themselves still have to know which to ask for?

GONNET: In the case of integration, Maple gives an answer that is, at this point, far superior to what most mathematicians would do, because Maple has algorithms, in particular the Risch algorithm, which will determine whether a function does not have an integral in a given setup. For example, if your function does not have an integral based on trigonometric functions, exponentials, and logarithms, this algorithm will tell you so. That's a very difficult answer for somebody doing it with paper and pencil. If you find an answer, it's okay; but if you don't find an answer, to say "there is no answer in any expression involving these special functions" is a very difficult thing to do. I don't think that most people know that there is such an algorithm. Maybe nowadays it's more common knowledge, but the algorithm is not so old. It's an algorithm of the last twenty five years or so.

HAIGH: That would give rise to another question, which would be the extent to which the needs of computerized projects like Maple have influenced research priorities within mathematics among people who might not be directly involved in making software. Are the needs of these systems having any influence on other people who might be coming up with these new algorithms or changing their perception of what might be an important area to work on?

GONNET: It's true that the mathematical thinking is slightly different than the algorithmic thinking. They are not completely different, but they are slightly different. For example, a mathematician would normally not be inclined to find a methodology to prove that a certain function doesn't have an integral. He or she might want to prove that a certain expression doesn't

have an integral, but will not be inclined to find a methodology to in general find that it doesn't have one. By the way, Risch is a mathematician, so in this case maybe a computer algebra person, but definitely not a system designer. He's a mathematician more than anything else.

Most of the algorithms for computer algebra and the algorithms that are used have come from a relatively small set of mathematicians/computer algebra people. I don't think that mathematicians at large are contributing to computer algebra. I think we have changed mathematics in many ways. Maybe not mathematical research, but we have changed mathematics in several ways. Just to give you an example, very early in the history of Maple, I'm talking about mid '80s, we started trying to solve the problems of the *American Mathematical Monthly*. I subscribe to the *Monthly*. You can see it over there. It comes with a section of mathematical problems which challenges mathematicians across the world. These are typically difficult problems that quite often have cute solutions. Some people work very hard, and then the best solution is published. Some problems don't even have a solution and go unsolved for very long times.

It's a very special but an active part of some mathematician's work—solving problems. Some problems are completely unreachable for Maple. To prove that some space is a Banach Space was my classical example. Maple doesn't even understand about those concepts to be able to prove anything. In the area of summation, asymptotic expansions, limits, and so on, Maple is quite powerful. It turns out that possibly about one out of ten problems were solved by Maple. So at some point I started interacting with the editors of the problem section, and they became aware that there were problems that were very difficult for humans but could be solved automatically. For a short period of time, they were sending me the problems before publication. Can Maple solve this problem? Because, if Maple can solve a given problem, clearly it's not an interesting problem for the community. I think that after that they got into the notion that there were some classes of problems which had appeared earlier in the *Monthly* that were challenging but were no longer challenging because those could be done automatically. Clearly something that can be done automatically is not challenging for a mathematician. It's not challenging for most people, right?

HAIGH: Do you think that more could have been accomplished if more mathematicians had been interested in working on this area, or do you think that there's been sufficient effort devoted to it within the mathematical community?

GONNET: Mathematics is a very large area. The mathematics that Maple does is a very small part of all of mathematics, a very, very small part. It is a large part of scientific computation, but it's a very small part of mathematics. As I said before, the problem of even getting these mathematical problems into a context that could be described in (symbolic) computational terms is very difficult or impossible.

So your question is difficult to answer, and I would say we have had very good cooperation from mathematicians, but that only a very small number of mathematicians have worked for computer algebra. That's okay because only a very, very small part of mathematics can be done by Maple or by computer algebra systems. It happens to be a very large portion of what scientific computation is all about, but it's a very small portion of mathematics as a whole.

One final random comment about the interaction between classical scientific computation and symbolic algebra. In my courses in scientific computation, I always make a speech, about the reliability of scientific computation. The reliability of scientific computation is a big topic. Why? Because typically you write a program, you compute for hours, you get an answer which is

typically a number or a set of numbers, but a small result. What guarantee do you have that this is correct? Well, the guarantee that you have that this is correct is that you have been a good programmer and that your data is correct and that you didn't make any mistakes. That's not a big guarantee. If you write a database program, you can store data, you can search it, and you can see whether you find it or not. If you write a payroll program, you can see whether the people come and complain they got the right salary or they didn't. In scientific computation if you simulate some property of some semiconductor, once you get the result, what guarantees do you have that this result is correct? In particular when it involves hundreds of hours of computing or something like that. It's very difficult to prove that these results are correct. Traditional software engineering practices to prove that your programs are correct basically fail. I think that symbolic computation has a lot to offer in this area, which has really not been exploited. Scientific computation will typically work the problem in a particular path and find an answer that is very difficult to compute. Of course computer algebra cannot do the same because, by nature, symbolic computation is a slower process. But what people don't realize is that quite often you can verify some of the properties of your answer back from the original problem.

Let me put an example of minimizing a function in many variables, which is a very common problem in lots of engineering problems. Suppose you have a complicated function. You want to optimize something. You want to find a minimum, maximum, whatever. Suppose you have hundreds of variables. You are going to run a complicated minimization process which is going to take a significant amount of computing. So you have an initial definition of a function. You have a lot of intermediate computation. You finally get a point where that is your minimum. In computer algebra you could not reproduce the same thing because you are not going to the same amount of computation, it will just take too long. But, if that point is a minimum, there are some properties that it has to satisfy. The partial derivative should be very close to zero. The Hessian should be positive definite, etc. You can now pose a problem in symbolic computation where typically you are closer to the original definition of the problem, where usually you can verify the original functions and test them and convince yourself that you have the right scenario and verify some of the properties of the minimum. If you made a mistake in the computation and you have the wrong function at some point, they will not verify. I think the bottom line of this of this point is that I think that computer algebra has a lot to offer to the reliability of scientific computation, which normally is a serious problem. I think that that aspect has not been explored to full satisfaction or to full profit of the people that are doing scientific computation.

HAIGH: Perhaps now you would like to talk a little bit about your involvement over the past 10 or 15 years with other companies.

GONNET: I would like to start with Joint Technology Corporation, which later merged with Aruna PLC in the UK. This is a company that is developing fast relational database engines, and all the research for the product was started out of a grant at the University of Waterloo. The topic of the grant was to incorporate or to merge relational database technology and text searching technology. This is a joint grant held by Frank Tompa, Paul Larson, and myself that started in the early '90s. Out of this project, one of the researchers hired by this grant was Mariano Consens, a graduate student from the University of Toronto. He was working with me in some aspects that I would describe as implementing relational databases using text searching algorithms. The research that we were doing proved to have very interesting results that we thought were very valuable. We decided to build a company around this research and around these results.

The company had a bit of a slow start. We hired a person to start coding a prototype, and for quite sometime that didn't go very fast, but at some point in the mid- to late-'90s we started a company in Canada that was called Joint Technology Corporation, JTC, and a product was produced. It's a very competitive area. Our product is very good, but we are competing against Oracle, against Sybase, against SQL Server. These are mature products backed by big companies. We were also hit to some extent by the bubble of the 2000s in that at some point people were very willing to spend huge amounts of capital in software, but that dried up very drastically. This is a product at the high end. This is a product that sells for hundreds of thousands of dollars and you don't expect to make many sales. You expect to have very few, and very precious customers.

HAIGH: I saw the names Freedom Intelligence and Aruna on your resume. Are those different names for the same thing?

GONNET: No. Freedom Intelligence was a marketing name for Joint Technology Corporation. The name Joint Technology was deemed not appropriate. And I don't deal with marketing; I let marketing do their thing. Aruna was a separate company a PLC in the UK. In 2002, Aruna PLC, and Joint Technology or Freedom Intelligence merged into a single company. The future is still uncertain for this company, so I cannot say more at this point.

HAIGH: What's the name of the product?

GONNET: The name of the product is FastPath, and it continues to be what we designed originally, a relational database which is geared for relatively static data (that is not updated continuously) and that you want to query in an ad hoc manner, and of course you want to query efficiently. Data that you add to every night or ever week or every hour, but not data that is transactional and you update every second. This covers the area of data warehouses, business intelligence, areas where people have a huge amounts of data, operational data or historical data, and they want to extract value, information, or make decisions based on this data.

HAIGH: So this isn't a tool that would plug into a standard database engine like Oracle? It would be a separate repository of data for reporting purposes?

GONNET: Right.

HAIGH: I would just have two follow-up questions on that then. The first one would be just in terms of your own personal role within the company, and roughly how much of your energy it's taken up over the past decade.

GONNET: This actually touches a bit of Maple, even though it's relational databases. I managed to weave my projects together every so often. My role in Joint Technology and Aruna has been certainly a non-management role. I have been interested in the technology from the beginning, and I have been part of the driving force behind the technology, but I have never been managing the company. I have played the role of chairman of the board, but I am not an active officer of the company, and I was never an active officer of the company.

On the technology side I worked originally, as I said, with a grant in studying the way of implementing relational databases based text-searching technology. But then later I became very interested in the query processing that is the way of transforming an SQL query into a plan (given the primitives that you have you can execute) to get an answer. This is what is normally called query processing, and query optimization is to select the plan that will execute the fastest, knowing the speeds of each of the primitives. I had some ideas about this how a query processor

should be implemented, and I found it very difficult to transmit it to the team of programmers that we had at the time. I decided that I was going to write a prototype in Maple.

HAIGH: By “we”, at this point do you mean the research group within Waterloo?

GONNET: No. This was the company in Waterloo. The Joint Technology Corporation in Waterloo. I started writing a prototype in Maple that would do SQL query processing. I got quite excited about it. It was really a lot of fun. And the prototype instead of being something that would show how to do things became a very live prototype, so much that it lasted for the whole history of the project. Even nowadays, the team that implements the relational database has two versions of the software: the Maple prototype, and the production version. Of course the Maple prototype is used every time that they need to make a change or develop new algorithms or whatever, they are written first in Maple, they are tested. It’s a nicer environment. And then they are written in C++. It also provides a tremendous amount of reliability because the two systems are written independently, have independent code bases, and we can verify that the results of one coincide with the results of the other. So it has provided a huge amount of reliability to the final software. This is in some sense another case of what I was mentioning before. In this case in a commercial application that Maple ends up being another way of testing things, not to do the extensive queries, but to verify that the software that is running for very long periods of time is correct. It’s a way of witnessing the software. It becomes a software engineering tool.

HAIGH: Am I right in thinking that this represents an evolution in the purpose of your involvement in this area, from originally producing a relational database that would be optimized for text retrieval into producing an extremely efficient query optimization system?

GONNET: Correct. This is an evolution of my original interest in algorithms that went into text searching algorithms that evolve into text searching, making relational databases, or searching relational databases. At some point the searching of relational databases needed query processing. By the way, there is a patent in the query processor that I think it’s the most valuable patent that the company has. The essential part of query processing, the way that we do it, is that we have this notion of symbolic computation behind it. We are able to transform the queries. For us, it’s completely natural to transform the queries according to mathematical formulas because our background is in computer algebra. It’s an area that has worked very nicely.

Web Pearls, I don't think it’s worth describing a lot about Web Pearls. We had a very nice idea with Web Pearls to produce learning material and testing material that would have a mathematical engine like Maple behind, and hence have the uniqueness that you can ask a question that the user may reply in different formats and you can process it mathematically, or you can generate problems randomly and they all make sense mathematically. Not just mathematical problems; even physical problems. We developed the software, but somehow at one point we were ahead of our time. Another problem we found is that high school teachers and college teachers are not ready to spend any money whatsoever for anything, or just cannot. The product has failed to fund itself, so I have been funding this company now for years and years and years. Last year I said, “This is it. I cannot fund it any longer. You either survive on your own or you don’t.” Now actually they are managing to survive, but I don’t know for how long they are going to survive. I have invested millions of dollars in this company. In both companies I actually have invested a significant amount of money. In JTC/Aruna and Web Pearls.

[Tape 6 of 6, side A]

HAIGH: Just to step back slightly from all the specific things that we've talked about, I wonder if I could ask you over the course of your professional career, what do you think would be your biggest regret, either in terms of something that you did or just in terms of the way the world reacted to something that you wish had gone differently?

GONNET: Well, I don't know if I can say regret. I don't think I have any big regrets. There were two instances which were unpleasant, and I wish I hadn't gotten into those. One is the breakup situation of the Maple company and how it evolved. I wish that I had selected my colleagues with more care. The second one was that I shouldn't have had any relations whatsoever with venture capitalists, because as I said before, there is nothing to be gained. They are all nice people, all very polite, all smiles. When there is some real money to be gained, they are ruthless. Those were unpleasant situations. In one case I ended having to go to the courts to get a settlement of what I thought was right, and the courts favored my side. But that's never a nice process; it's always a very damaging, time-consuming, and time that you end up hating having to do it. On the other hand, if I had to do it again, would I do it again? Probably I would do it again because I would say, "Well, I need to be more careful." Maybe I still didn't learn the lesson.

Many times people have asked me, "Why did you let Maple get out of your hands? Why did this happen to you? Why did you let Maple get out of your hands?" The answer is probably that I never acted as if I owned Maple. I always wanted it to be a collective project. Maybe this is a sad conclusion that maybe you shouldn't do that. Wolfram is obviously more successful by having an iron fist and controlling everything. Actually, Wolfram started in a much more democratic situation with several colleagues with which he shared part of his company and turned it into a situation where he owns basically everything and calls all the shots. Maybe I should have done the same, and maybe Maple would then be competing with Mathematica now or it would be ahead of Mathematica. I have always had the idea that there was more value in having a group of colleagues that we would all be working towards a common goal, and I overlooked the fact that maybe some of those colleagues at some point will turn around and stab me in the back. That's about the Maple project.

In the case of Open Text, I don't have many regrets. Open Text was a good project, a very satisfactory project from the technical side, a fun project. The new OED and the initial times in Open Text were good technical times and fun times also. Economically it was not bad. At some point I owned 37% of Open Text, of course, that ended up being diluted when we got further investment and so on, but I had a very significant investment in Open Text personally. I regret having had to battle venture capitalists, but I guess that I am not the only one in the world that has had those problems.

HAIGH: The more positive reverse of that. As you look back over your career, what would you say are the one or two achievements that you're proudest of?

GONNET: That's a good one. I think I'm very proud of Maple as a whole. I'm very proud of all the small and big architectural decisions that went into that, and proud even of the mistakes that I have made.

I think that the prototype of the query processor for SQL was also very, very rewarding. I think also PAT, the fast text searching for the OED project and later for Open Text was also very rewarding. The case of PAT, it's a very peculiar piece of software. I had a graduate student in Waterloo by the name of Nivio Ziviani. He was from Brazil. Nivio had worked on similar ideas and was my PhD student while I was working on text searching. He always thought that writing

a program that will do text searching would be a very good thing and was always pushing me to write such software, and I never had the time. One day I was going to a conference in Chile in South America, and I had to change planes in New York. There was a big storm, and I got stranded as I arrived very late from Toronto, lost my plane to go to Santiago, was stranded in New York for 24 hours, didn't have absolutely anything to do, and actually had to be close to the airport. I was trying to catch some other flight. It was like you have an empty slate, no luggage, nothing, just your briefcase, nowhere to go, 24 hours ahead of you, so I decided to start writing the text searching program. I actually wrote a lot of it that day. That was the way that the text searching software started. It had a very peculiar start in a hotel near JFK Airport.

HAIGH: Unless you have anything else that you would like to say or any comments on what your priorities will be for the years to come, then that will be the conclusion.

GONNET: I think I should say a few things about some of my colleagues in this area. As I mentioned before, several people have been role models for me in this discipline. Cleve Moler, I consider a friend and a role model. I have a tremendous respect. I think Gene Golub is an example of a great scientist, and he's also a role model for me. I think that Don Knuth is also a role model for me. We mentioned yesterday William (Velvel) Kahan, and I think very highly of him too. I also think of my colleagues here in Switzerland, in particular Walter Gander has been extremely helpful and supportive. I also think that the times that I worked with Frank Tompa in Waterloo were super, and we did great work together. I also think the times that we were with Keith Geddes before 1994 were really good times, very positive times, and we achieved great things. Really you never work alone—whenever there is something good happening it's because you're surrounded with good people too. That's a fact of life.

HAIGH: Thank you very much for taking part in the interview and your time over the last three days.

GONNET: You're very welcome. Thank you.