News from New Zealand

BY

C. S. Calude

Department of Computer Science, University of Auckland
Auckland, New Zealand
cristian@cs.auckland.ac.nz

1 Scientific and Community News

0. The 14th International Conference on *Unconventional Computation & Natural Computation* will be held at the University of Auckland, New Zealand, 31 August – 4 September, 2015, http://www.cs.auckland.ac.nz/research/conferences/ucnc2015.

1. The latest CDMTCS research reports are (https://www.cs.auckland.ac.nz/research/groups/CDMTCS/researchreports/):

462. C.S. Calude and D. Desfontaines. Universality and Almost Decidability, 05/2014

463. C.S. Calude and D. Desfontaines. Anytime Algorithms for Non-Ending Computations, 06/2014

464. R. Nicolescu and D. Vaida. Indexed Grammars, ET0L Systems and Programming Languages: A Tribute to Alexandru Mateescu, 07/2014

465. M.J. Dinneen and Y.-B. Kim. Deterministic Transistion P Systems Modeled as Register Machines, 07/2014

466. M.J. Dinneen and Y.-B. Kim. Simulation of Functional Register Machines using Active P Systems, 07/2014

467. M.J. Dinneen and Y.-B. Kim. Using Membrane Systems to Solve the Bounded Fanout Broadcast Problem, 07/2014

468. S. Link and H. Prade . Possibilistic Functional Dependencies and Their Relationship to Possibility Theory, 08/2014

469. S. Link and H. Prade. Relational Database Schema Design for Uncertain Data, 08/2014

2 A Dialogue with Kurt Mehlhorn about Theory to LEDA and Algorithm Engineering

Kurt Mehlhorn `http://www.mpi-inf.mpg.de/~mehlhorn` *is the head the Algorithms and Complexity Department of the Max-Planck-Institut für Informatik, Saarbrücken, a professor at Universität des Saarlandes, a co-director of Indo Max Planck Center for Computer Science, the chairman of the Scientific Board, IST Austria, and a member of the Joint Advisory Board, Carnegie Mellon, Qatar.*

He was educated in computer science and mathematics at TU München and obtained his PhD at Cornell University, Ithaca, USA with the Dissertation "Polynomial and Abstract Subrecursive Classes" (at the age of 25). He has published more than 250 papers in various areas of Informatics as well as the influential books Algorithms and Data Structures—The Basic Tool Box, The LEDA Book, Data Structures and Algorithms *(three volumes) and* Programming Languages.

Professor Mehlhorn was awarded 17 distinctions (1986–2013) including elected Fellow of ACM (1999), elected member of the Academia Europaea (1995), Berlin-Brandenburgische Akademie der Wissenschaften (2001), Deutsche Akademie der Naturforscher Leopoldina (2004), Acatech, Konvent Technikwissenschaften (2004), Bayerische Akademie der Wissenschaften (2012), and United States Academy of Engineering (2014), and many prizes including Leibniz-Award (1986), Karl-Heinz-Beckurts-Award (1994), Konrad-Zuse-Medal (1995), EATCS-Award (2010), ACM Paris Kanellakis Theory and Practice Award (2011), Khwarizmi International Award (2013), Erasmus Medal of the Academia Europaea, and honorary doctorates from the Otto-von-Guericke Universität (2002), the University of Waterloo, Canada (2006) and Aarhus University, Denmark (2008).

He has 24 students and 93 scientific descendants.

CC: Please reminiscence about your studies in München (1968–1971) in both computer science and mathematics.

KM: It was clear to me since age 14 or so that I wanted to study mathematics; not, that I knew what a mathematician would do for a living. I did not know any mathematician besides my high school teachers. However, the subject fascinated me early on. I started to study math at the Technical University in Munich in 1968. This was the year, when Informatik (computer science) was introduced as a new field of study at the Technical University in Munich and 6 other German universities. A friend of mine convinced me to also attend the introductory Computer Science lectures. I had little idea what computer science was about. I had read about computers in popular science magazines, I knew them from pictures, but I had never experienced a program at work. The course was taught by F. L. Bauer, a German computer science pioneer.

So, I started with mathematics, physics, and computer science. I dropped physics after the end of the second year. I remember the lectures in mathematics and computer science to be very different. Whereas, the math lectures were polished, the computer science lectures were rough, and the content was of quite uneven difficulty. It was obvious that the instructor taught most of the material for the first time. The course also introduced us to programming. The programming language, F. L. Bauer used in class, was Algol 68. I tried to read the Algol 68 Report, which is a 168 page definition of the language. This report is one the most difficult document, I have ever tried to read. I never made it through.

In the lab, we used Algol 60 and later Euler as our programming languages. We ran our programs on the PERM (Programmierbare Elektronische Rechenanlage München) and later on a Telefunken TR4. I was fascinated by programming from the very beginning. It is an act of creation. Programming creates an artifact, the program, that "lives". It consumes inputs and returns outputs. In order to program, one has to understand a problem so well that one is able to teach a machine to solve it. And amazingly, the machine does it faster and more reliably than the creator himself.

CC: Then you obtained your PhD in computational complexity from Cornell University with Professor Robert Constable as adviser. Why did you move to US? How was computer science at Cornell in early '70s?

KM: Back then, German students had to take a major exam at the end of the second year. I did well; so my professors organized a one-year scholarship to the US for me and two of my fellow students. I was sent to Cornell and entered Cornell in 1971 as a special student. Cornell only had a graduate program in CS at that time. I took the courses in the PhD program, did well, and so the department offered me a scholarship for PhD-studies. This was the summer of 1972. We had long planned that my then girl friend, now wife, would came to the US in the summer of 72 for a trip to the West Coast. We went on the tour and we discussed our future. We had been intensive letter writers for the past year and did not want

to continue that way. My girl friend agreed to move to the US. We then briefly returned to Germany in the fall and got married.

Back then, Cornell was a small department, but with excellent faculty, e.g., Juris Hartmanis, John Hopcroft, Robert Tarjan, Bob Constable, Jerry Salton, and David Gries. I was particularly fascinated by Bob Constable's lectures and so wanted him to become my advisor. I remember that being afraid that he would say 'no', it took me days to build up the courage and actually ask him to become my advisor. Luckily, he said 'yes'.

Bob's interests spanned complexity, constructive mathematics, logic, and semantics, and so I was introduced to all four subjects. I also received a profound introduction to algorithms by Hopcroft and Tarjan. The fact that I was recently able to publish on a framework for verification of certifying computations (Journal of Automatic Reasoning) goes back to Bob's influence.

CC: As a student I have read with great interest your first paper "On the size of sets of computable functions, in *Proceedings of the 14th IEEE Symposium on Automata and Switching Theory*, 190–196, 1973". Later, I have used several times a constructive form of the Baire category theorem to measure the size of various sets of objects in discrete spaces which you introduced in that paper.

KM: I am flattered. At that time, many researchers in complexity theory studied recursion theory, the mathematical study of degrees of non-computability. We explored which of the notions would carry over to the study of degrees of computability. I had come across a paper that used Baire category theory in recursion theory, and I transferred those ideas to the domain of computable functions.

CC: You have important contributions in many areas of theoretical computer science: in computational complexity, data structures, computational geometry, computer algebra, parallel computing, VLSI design, combinatorial optimisation, and graph algorithms. What was the motivation for such a variety?

KM: This was never a conscious decision. It just happened. I talk to colleagues, I listen to talks, I read, and when something catches my attention, I start thinking about it.

Of course, some of my work follows a long-term plan. I have recently published some papers on isolating real roots of univariate polynomials. The fact that I worked on this problem, can be traced back more than 30 years. I developed an interest in computational geometry soon after the field was introduced by Shamos and Preparata. The third volume of my 1984 book on Data Structures and Algorithms treats Computational Geometry. It never occurred to me at that time, that the implementation of (geometric) algorithms might be difficult. The book starts with the sentence that the machine model used in computational geometry is the Real-RAM, a RAM that can compute with real numbers. It did not occur to me that this machine model is far away from real machines and that the gap to real

machines might be non-trivial to bridge. I asked students of mine to implement some of the algorithms as part of their master's degree. A very good student (he later got a PhD and is now a leading figure in a software company) implemented Voronoi diagrams of line segments. It was a beautiful piece of work, but it worked only for some examples. I tried myself, but could not do it any better.

What had gone wrong? We had implemented the Real-RAM by substituting floating point arithmetic for real arithmetic. The round-off errors killed us. Geometric programs branch on geometric predicates, e.g., does a point lie left of, on, or right of a line. The outcomes correspond to the sign of an arithmetic expression in the coordinates of the points and the line parameters. If the sign is computed incorrectly, a wrong branch is taken, and the computation may go astray. Let me give a simple example. Consider a polygon in the plane, a point outside the polygon, and assume that the edges of the polygon are oriented counter-clockwise. Then the point sees a contiguous set of edges of the polygon, but it does not see all of them, where a point sees an edge if it lies to its right. Therefore one can find all visible edges by finding one and then walking in clockwise and counter-clockwise direction until one reaches an invisible edge. Due to round-off errors it may happen, that the point sees all edges of the polgygon. Then the strategy just outlined will never stop. It is not too hard to find concrete points, where exactly this happens. The paper "Classroom Examples of Robustness Problems in Computational Geometry" contains many other illustrative examples.

This experience spurred my interest in laying theoretical foundations for the efficient and correct implementation of geometric algorithms. In the 90's we and others did so for linear geometry: rational arithmetic, floating point filters, LEDA geometry, and CGAL. Let me explain the idea of floating point filters. The evaluation of the arithmetic expression underlying a geometric predicate usually returns the correct sign; only in nearly degenerate cases, an incorrect sign may be reported. In these cases, the value of the expression is close to zero. The approach is now as follows. We first analyze the arithmetic expression and prove a theorem of the form: if the absolute value of the expression is larger than some epsilon then the floating point evaluation gives the correct sign; the value of epsilon comes out of the analysis. If the value is smaller than epsilon, the sign may or may not be correct. In this case, we reevaluate using exact arithmetic. In this way, we always compute the correct sign, but pay the higher cost of exact arithmetic only when needed. Experience shows that the necessity for exact arithmetic is rare. In this way, we arrived at correct and efficient implementations of the algorithms for linear geometry.

At the beginning of this century, we moved to geometry with curved objects. The problem of providing the appropriate Real-RAM becomes much harder. Notice that the intersection point of two lines with integral coefficients have rational coordinates. However, already the intersection of a quadric and a line may

have non-rational coordinates. Thus, the reliable implementation of geometry with curved objects requires algebraic arithmetic. A key subroutine is the computation of the real roots of a univariate polynomial. More precisely, one wants to compute disjoint intervals with rational endpoints, one for each real root, such that each interval contains exactly one real root. This is a fundamental problem in computational algebra. Algorithms for polynomials with integer coefficients were available, but we needed an algorithm for polynomials with arbitrary real coefficients, where a real is given as a dyadic number with a potentially infinite number of bits after the binary point. The algorithm would be allowed to ask for additional bits when needed. There were algorithms by Schönhage and Pan which could handle the situation, but nobody had ever implemented them. I started working on the problem and got Arno Eigenwillig, Michael Kerber, and Michael Sagraloff interested. I had never worked in computer algebra before. We soon had a first success and now have a simple algorithm that matches the asymptotic performance of Pan's algorithm. We were lucky in the sense, that the solution of the problem did not require any deep knowledge in computer algebra.

CC: Can you discuss one of your theoretical result you like most?

KM: It is actually one of my early results, obtained in 1981 and 1982 together with Scott Huddlestone. We studied balanced trees. After an insertion or deletion balanced trees need to be rebalanced by rebalancing operations. It was known that in the worst case a logarithmic number of rebalancing operations is necessary, and it was also known through experiments, that the average case is much better. Nobody knew how to analyse the average case. We found that a stronger theorem is much easier to prove. We studied two-four trees and showed that the total number of rebalancing operations required for any sequence of insertions and deletions is linear in the length of the sequence, i.e., the amortised number of rebalancing operations per insertion/deletion is constant. For the analysis, we associated a bank account with the data structure. Such accounts are now called potential functions. Our analysis was one of the first examples of an amortised analysis. The theorem is now taught in many data structure courses and the proof method has numerous applications. Previously, Norbert Blum and I had proved a similar result for weight-balanced trees.

CC: You are one of the developers of LEDA, the Library of Efficient Data types and Algorithms. Please explain the goal of the project and how LEDA is currently used.

KM: In the mid 80s, Thomas Lengauer and I started a project on VLSI-design. As part of the project, we wanted to build a system that translated high-level descriptions of VLSI-circuits into compact layouts. The implementation effort encompassed Postdocs, PhD, and Master's students. Progress was very slow. When analysing why, we identified that one of the reasons was that several data struc-

tures and algorithms had been implemented several times. In the late 80s, there was no easy way of reusing implementations and there was no repository of good implementations.

Based on this experience, Stefan Näher and I started the LEDA-project in 1988. We first reported about the project at MFCS 89 and ICALP 90. The reaction by our colleagues were luke-warm. The goal was to provide fundamental data structures and algorithms as easy-to-use, efficient, and correct modules. We spent the first six months of the project on selecting the programming language and defining the interface of the priority queue data structure. We implemented priority queues and Dijkstra's algorithm. We wanted a specification of the priority queue data structure that would work in Dijkstra's algorithm without prior knowledge that it would be used in this algorithm. We tried C++, Modula, Eifel, C, and a few other languages, and decided that only C++ with its template mechanism would allow us to reach our goals. C++ was still in early infancy at that time. In Dijkstra's algorithm one associates distances with vertices, from time to time decreases the distance of a vertex, and in each iteration needs to select the node with minimal distance. We found the interface descriptions in the algorithm textbooks (including my own) not very useful, since the insert as well as the decrease-key operation would have a vertex as an argument, a violation of the goal, that the data structure needs to know nothing about its later use. We decided that the argument of an insert operation would be a priority and the insert would return a handle to the object storing the priority. In LEDA, we use the word item instead of handle. In the graph data structure, we would associate the item returned by the insert with the appropriate node. The decrease-key operation has two arguments: a handle and a new priority. In this way, we had completely separated the data structure and the graph algorithm. The item concept is used throughout LEDA.

The project developed nicely and we soon had a siseable user community. However, by 1994, we were also facing some serious problems. First, none of our geometry algorithms worked for all inputs. All of them would break on some or even most inputs. This was due to the naive use of floating point arithmetic as a substitute for real arithmetic. It spurred my work on exact geometric computation mentioned above. A first paper appeared in the ACM Geometry Conference in 1995.

Second, even the implementations of some combinatorial algorithms were incorrect. For example, the first version of the planarity test, declared some planar graphs non-planar. The program had been written by one of my Master's students. One day, Stefan Näher came to my office holding a letter in his hand. The letter contained the planar drawing of a graph that our system had declared non-planar. Stefan said: "Kurt, it was one of your students, who implemented this program". It was clear, that Stefan expected me to fix the mistake. I did. More importantly, Stefan and I started to discuss how we could make our implementations more re-

liable. We came to the conclusion, that, if possible, all programs in LEDA should be certifying, i.e., for every instance compute a witness (proof) that the output is correct for this particular instance. For example, the planarity test now also outputs a planar drawing whenever it declares a graph planar, and it outputs a Kuratowski subgraph when it declares a graph non-planar. Over a period of five years, we succeeded in making most of our programs certifying. The LEDA book of 1999 and a survey article of 2011 give more details.

Certifying algorithms became a theme for me. Whenever I develop a new algorithm, I ask myself whether it can be made certifying. Certifying algorithms come with a checker program, that inspects the witness and verifies its validity. In other words, the checker validates the work on the main program. "And, who validates the work of the checker?" I have been often asked. I used to answer that the checkers are very simple programs and hence their correctness is self-evident. I now reply that the checkers are so simple that they are amenable to formal verification. Over the last two years, we have formally proved the correctness of several witness theorems and of the corresponding checkers. A witness theorem states that a witness of a certain type proves the correctness of an output, e.g., a Kuratowski subgraph proves non-planarity of a graph.

Stefan Näher, Christian Uhrig, and I founded Algorithmic Solutions in the mid 90s. It markets LEDA, develops it further, and offers algorithmic consulting. We have a solid academic and industrial customer base.

I see LEDA also as a role model for other library projects such as CGAL and LEDA-XXL.

CC: What is "algorithm engineering" and what are your contributions in this area?

KM: There are many definitions of algorithm engineering. I'll give you my personal one. Treat programs as first class citizens in algorithms research and not as an afterthought. A number of research directions follow immediately from this. Care about constant factors in the running times of programs and also algorithms. Study the implementation process. Can it be improved by algorithmic means? How to go from a correct algorithm to a correct program? Study machine abstractions. Are they a useful abstraction of reality and do they allow for meaningful predictions? If a prediction made by a model conflicts with experimental evidence, try to find the reason and then either refine the prediction or revise the model. Care not only about the worst case, but also about the typical case and the best case. What are typical inputs? Can one characterise them?

Algorithm engineering is not only a sub-discipline of algorithms research. More importantly, it is a mind set. I frequently say that I am a mathematician in the morning and an engineer in the afternoon. I like to do math, in particular, math that can be made operational. I also like to turn math into systems.

It gives me great satisfaction to see my systems used. The experimentation with systems generates well-motivated new research problems and sometimes suggests theorem.

CC: You have played an important role in the establishment of several research centers for computer science in Germany. What was the driving force?

KM: I helped establishing the Dagstuhl center and also the Max Planck Institute for Computer Science. Dagstuhl, because it was decided to locate it near Saarbrücken (Günther Hotz and Heinz Schwärtzel were the driving forces behind this decision) and I was chairman of the computer science department at that time. The Max Planck Institute, because Harald Ganzinger and I were selected to be the founding directors of the institute.

I have served in many other functions, e.g., as a member of the senate of the German Research Foundation, vice-president of the Max Planck Society, and member of the senate of the Max Planck Society. I believe that every individual owes part of his/her time to the societies he/she is a member of; my yardstick is about 10 to 20 percent. I love to work in organisations that function well and I am willing to invest time in keeping them that way.

CC: You are serving in the editorial boards of many international journals. New models of scientific publications have recently appeared. How do you see the future of scientific publication?

KM: This is a very difficult question. I strongly believe that all results of publicly funded research should be openly accessible. I do not conclude from this that all journals and conference proceedings must be open access. For example, the theory community has realised open access through the arXiv and institutional and personal repositories to a large extent, although most journals and conference proceedings in the field are not open access. Nevertheless, I believe that scientific publishing will move to the author-pays open access model over the next 10 years. Of course, author-pays has to be implemented as "the institution or funding agency of the authors pays".

Publishing is also about long-term archiving. Academic institutions and also publishing houses are such institutions. I am sceptical when well-intentioned individuals create a new journal without the backing of an institution that can guarantee sustainability.

Publishers (both for-profit and not-for-profit) provide useful services. For example, electronic access to journals was widely introduced at around 2000. At that time, this required substantial investments which only major publishers could make easily. Publishers will play an important role also in the future. The switch to electronic distribution of content has lead to great increases of productivity. However, some of the for-profit publishers have forgotten to share the resulting savings with academia, and, as a consequence, the profit rates of some publishing

houses have reached unfair heights. In their own interest, publishing houses must return to a fair cooperation with academia.

CC: How do you manage to juggle between so many jobs in different countries?

KM: I try to follow some simple principles.

I avoid multi-tasking. I set aside time for particular tasks and then concentrate on them. For example, when I was writing my 1984 books and the LEDA book, I would work on the book every work day from 8:00 am to 12:00 pm. I would not accept phone calls or interruptions by students during this time. Now, the 8am to 12pm slot is reserved for reading, thinking and writing. The no-interruption rule still holds.

I clean my desk completely every evening when I leave my office, so that I can start with an empty desk the next morning.

When I accept a new responsibility, I decide, what I am going to give up for it. For example, when I became vice-president of the Max Planck Society in 2002 (for a 6 year term), I resigned as editor of Algorithmica, Information and Computation, SIAM Journal of Computing, Journal of Discrete and Computational Geometry, International Journal of Computational Geometry & Applications, and Computing.

And most importantly, I am supported by many people in what I do, in particular, my co-workers, my students, and then administrative staff in the institute and the department. Cooperation and delegation are very important.

CC: How do you treat your own mistakes?

KM: Of course, I try to avoid them in the first place. Also, I easily forgive others for making mistakes. I treat myself no differently.

Let me be serious. I restrict your question to mistakes that I make as a scientist. I also make mistakes as a manager and these mistakes are a different story.

Scientific mistakes are, in principle, easy to deal with. Dealing with them might be a lot of work, but it is usually clear what one has to do. If one finds a mistake in a proof, one tries to fix it. If one cannot fix it and the result is already published, one admits the mistake. If one makes a mistake in a lecture, one corrects it in the next lecture or in a hand-out.

I have actually profited a lot from "scientific mistakes". All my work on certifying algorithms and reliable computational geometry was spurred by mistakes. An incorrect implementation is a mistake. Most of the time it is a mistake made by the programmer, but sometimes it leads to something deeper. Namely, if the mistake points to a deficiency in the state of the art. Then one has to try to advance the state of the art. If one succeeds, the mistake turns into a fortunate event. This has happened to me several times.

CC: Many thanks.