Passion for the missing argument

Wim H. Hesselink

21 February 2011

Ladies and gentlemen,

Some things are true. You can see that immediately. But why is it true? Can you prove it? Do you know it by intuition? Or is it just a matter of belief?

The same holds for computer programs. Some meet their specifcation. You may believe it. You can test it. If you want to be sure, however, then you need a proof.

I have chosen this problem area as the theme for this lecture, which you can also regard as a selective account of my working life up to my 65th birthday which falls today.

What is a proof? A proof consists of a sequence of arguments. If one argument is missing the proof may collapse as a house of cards. Hence the title of this lecture:

Passion for the missing argument

1 To begin: heroes from the past

Ever since, in the third century BC, the Greek mathematician Euclid systematically developed geometry, we know: when you want to prove things in mathematics, you need axioms to start with.

The same holds for arithmetic. The favorite axiom system for arithmetic is due to Peano (1889). In 1931, however, Kurt Gödel proved that every axiom system for arithmetic is incomplete in the sense that there are true statements of arithmetic that cannot be proved with this axiom system [15, 3].

In line with this thought, Alan Turing [18] invented in 1936 the Turing machine as a conceptual model of the *computer* (in those days, a computer was a person who performed a calculation according to a given recipe). Turing proved that there are problems for which there exists a recipe (computer program) that, in all cases that the answer is "yes" will eventually print "yes", while there is no recipe that, in all cases that the answer is "no" will eventually print "no". In this way, we know already 75 years that there are things that a computer can **not** do.

This was pure research, but nevertheless it has resulted in the computer with which we **can** do unbelievably many things.

2 The myth of Sisyphus

Barely a year ago, Peter van Emde Boas retired as a reader in Mathematical Computer Science at the University of Amsterdam. In his valedictory address he developed a one-to-one correspondence between his research projects and the labours of Heracles, the hero from Greek mythology. When I heard this comparison, I thought I would rather compare my activities with those of Sisyphus. Sisyphus, former king of the Greek city Ephyra, had often outwitted the gods, and had therefore been condemned in the underworld to eternally try and roll a block of stone in vain over the summit of a hill.

It is a familiar feeling for the experienced teacher: you try to teach something to your students, you have done so last year, and now you can start all over again.

3 How it started for me

Between 1970 and 1975, I did my PhD in Utrecht with prof. Springer in the algebraic geometry. This is a vast field. I then suppressed my passion for the missing argument. When I wanted to achieve anything new in that field, I had to build on what was proved by other people, and I should not try to verify every proof myself.

The summit of my mathematical career was in the fall of 1979 the publication of a paper [9] of mine in the Inventiones, a leading mathematical journal. In January 1980, however, our third child Mark Hessel died before he was a month old. This loss touched me so much that I lost the connection with the research front in my branch of mathematics. This gave rise to a career switch.

In 1982/1983, I followed courses in computer science to learn writing computer programs for solving an algebraic classification problem. I also mediated, as the chairman of the educational committee of our subfaculty, in a conflict about the course in Advanced Programming, which I had followed the same year. The students argued that the course required too much work.

The following year, this course should have been given by the newly appointed professor Bron. A few month after his appointment, however, it turned that he was unable to give the course. Out of desperation, they asked me who had taken the course the previous year to give it this time (1984). I accepted the challenge. It was hard work to keep ahead of the students, but it was highly satisfactory. The students were much more motivated for the course than for the mathematics courses I had experience with. They wanted to learn because they wanted to be able to do the programming assignments.

Although my students had no problem with it, the course included material about abstract data types that I did not like. In this simpler setting, I could give in to my passion for the missing argument. It became my first publication in computer science [10].

4 The transition

I officially moved from mathematics to computer science in 1985. I overcame my hesitation because of the enthousiasm of Jan van de Snepscheut, who had come here as a new professor in computer science in 1984. The switch was sealed with a sabbatical year in 1986/1987 at the University of Texas at Austin with the well-known Dutch computer scientist Edsger W. Dijkstra, who was Jan's thesis advisor.

July 1985, I first met Dijkstra at his home in Nuenen. In this interview, I tried to let Dijkstra divide computer science in subdisciplines. He was not willing to do this. For him, computer science was indivisible. The central question of computer science was to control the complexity of our artefacts.

In the year that I was in Austin, Dijkstra gave an honours course "mathematical methodology" for undergraduates. I took the course. It was a strange experience for me with my doctorate in mathematics. Dijkstra did not know much of mathematics, but he thought as a mathematician. A central point was to avoid or postpone case distinctions.

My weekly highlight that year was Dijkstra's Tuesday Afternoon Club, with Dijkstra, Jay Misra, Tony Hoare (also at Austin that year), and various others. I remember, e.g., how Amir Pnueli had to defend his temporal logic against the combined scepsis of Dijkstra and Hoare.

5 Programming methodology

Between 1975 and 1990, Dijkstra and his collaborators in Eindhoven and Austin developed a technique to *derive* algorithms or programs from specifications. I have learned the technique from Jan van de Snepscheut. I can do it only for sequential programs, but then it can be very effective.

When Jan went to Caltech, I developed a course in Program Correctness in which I codified the knowledge of program derivation. I extended it with recursive procedures with parameters as in the programming language Pascal. The careful semantical underpinning of this treatment has cost me (or yielded me) several scientific publications [11, 12, 13].

For more than 20 years, now, the technique of program derivation has been taught to our students in the first or second year. Passion for the missing argument has become our teaching goal. Indeed, the computer programs in this course are difficult enough to be wrong when they are not developed correctly. On the other hand, developing them correctly can be taught and learned.

5.1 Euclidean distance transformation

One example of the effectiveness of program derivation: around 1997, one afternoon, Arnold Meijster entered my office because he needed a program for efficient calculation of the Euclidean distance transformation. For now, you don't need to know what this is. Suffice to say that the concept was introduced around 1965, that an efficient approximating algorithm had been found in 1980, but that an efficient exact algorithm was unknown to us in 1997. I could not make it that afternoon. In fact, Arnold had asked me the same thing some months before, and I had not been able to do it that time. Yet, one never knows.

The same evening I started again with it, and now solved the problem with the technique of program derivation. The next morning, it turned out that after his visit to me, Arnold had also visited my PhD student Rutger Dijkstra. Rutger had solved the problem the same evening, with the same methods. Later it turned out that we were two years late. Efficient exact algorithms for the Euclidean distance transform had been published in 1995. For this reason, we only published our algorithm in conference proceedings [16], and withdrew a submission to the important journal PAMI. Some years later, we saw to our dismay that another group of authors had succeeded in publishing their version of such an algorithm in PAMI.

6 Management and organisation

In management circles, people are rarely interested in complete arguments. I have therefore seldom shown my passion for missing arguments there.

Yet, there was passion. Passion for quality. Quality of education and research, quality of teachers, quality of labour relations, quality of the relationship between students and staff, and quality as a teaching goal for computer science. Out of passion for quality, I have rolled with Sisyphus many a boulder up hill.

6.1 A first visiting committee

In 1979/1980, I was secretary of the board of the subfaculty of Mathematics, with Hendrik Hoogstraten as chairman. Because the faculty of Natural Sciences had no idea what was happening in mathematics, we were subjected in January 1980 to a so-called "visiting committee", an administrative innovation that has been used later on a much wider scale.

The opinion of the committee was very positive. Striking was the recommendation to take Computer Science out of the subfaculty, and to make it a separate group within the faculty. This has happened indeed. It had the good result that the necessary growth of computer science could be realised at the expense not only of mathematics. The decision is reversed later, but by that time the air was clear again.

6.2 Ups and downs of computer science

After my return from Texas in 1987, much had changed. Roland Backhouse had come as a new professor. Jan and Roland had taken the initiative to organize a scientific conference for the 375th anniversary of the university. It became "Mathematics of Program Construction", June 1989 [19]. The conference was a success: it was the beginning of a sequence of until now 10 MPC conferences.

In the mean time, however, Jan van de Snepscheut had become so frustrated by what he felt as a lack of collaboration higher up in the university, that in September 1989 he left for Caltech. His departure was a great loss for computer science in Groningen. He was an inspiring leader and an enthusiastic teacher and researcher.

In september 1990, Roland Backhouse also left, to Eindhoven University. The department was more or less back to **square one**. We had become frighteningly small.

6.3 Forward computer science

The department was strengthened in 1991 by the new professors Nicolai Petkov and Gerard Renardel, followed in 1993 by Ben Spaanenburg. After my appointment as a professor in 1994, I became chairman of the department. Looking back now, it was a good time. Not without problems, of course, but we had enough staff and good relations between staff and students to offer a good education. There was a more or less democratic structure in which I could work well. Research was the responsibility of the IWI, I'll come back to this later.

My function as chairman lasted to 1997. Then the departments (vakgroepen) were abolished and a vertical management structure was introduced. Research and education were separated. Research was to be organized by the local research school, and education by educational institutes. This change made me very unhappy. Not so much because I had lost my influence, but rather because the communication had disappeared. I did not know any longer what was happening, and nobody seemed to know. The old structures for communication had been removed, and new structures for this had yet to be invented and installed.

6.4 Research schools

The Dutch idea of research schools dates from 1991. Around 1995, things became serious. Nobody knew the consequences, but the idea was that there could not be any research outside the research schools. I therefore became a member of the research school IPA, which was created in 1995. The IPA was (and is) especially useful for the participating PhD students, but also for me it was a good opportunity for contacts with colleagues in the country and to participate in inspiring conferences.

The same year, 1995, in Groningen, the research institute for mathematics and computer science IWI was established. There were never any problems between these national and local forms of collaboration.

6.5 Committee Van Lint

The research review of 1997 went so bad for the IWI that, in the fall of 1998, the Board of the University installed a committee Van Lint for advice on how to proceed. Of course, the committee needed some time to investigate and to come with recommendations. With hindsight, it was only one year, but in my remembrance it is longer than the three years of my chairmanship. There was great uncertainty within the staff, and I could do nothing about it, because I lacked information myself.

The report of the committee was released in June 1999. It contained a dozen of recommendations. The highlight perhaps was the recommendation to strengthen the *esprit de corps* in the IWI. In order to strengthen research it was necessary to attract new talent, but only after rigorous selection. Researchers could only be admitted as *fellow* to the IWI based on quantitative indicators. Researchers that were not admitted to the IWI, should retain very limited opportunities to do research. They would have to do more teaching. I'll come back to the consequences of this.

6.6 Educational director

In the year 2005, I was educational director for computer science. We felt threatened by the upcoming educational review of 2006. One of the requirements we had to reckon with was that our courses, including those in the bachelor phase, had to be given by staff that was involved in research. This of course was in conflict with the decision to exclude teachers from research involvement.

Apart from this, at the end of 2005, I became involved in a conflict. Because I did not receive the recognition I felt entitled to as a guardian of the quality of our education, I resigned.

It was for me, and not only for me, very painful. Jos Roerdink, helped by several others, has performed a miracle in rewriting the self-evaluation in such a way that we came successfully through the educational review 2006. Gerard Renardel succeeded me as educational director.

Since those days, most of the teachers not admitted to the IWI have left the department, partly for other jobs within the university. We are therefore now in a situation that important courses in the bachelor program are given by external staff, hired for the occasion (at the moment I am one of these). Indeed, the courses have to be given and the staff with research involvement is too small, and has too much work in education and research, as well as with the writing of research proposals that have a very low probability of yielding money.

6.7 Exam committee

I don't remember exactly when I became a member of the exam committee. It must have been before 2001. In August 2001, there was a student who appealed because of the grade $6\frac{1}{2}$ for his final thesis. As the only member of the committee that was not on holiday I had to refer him to the Court of Appeal for the Exams of the University. At the same time, we had to start a mediation process. In the end, this mediation succeeded, and the grade $6\frac{1}{2}$ was retained.

Matters of appeal are the most interesting matters for the exam committee. In those days, most of the daily work was to decide whether a student who had obtained results under continuously changing educational programs in the end had obtained enough results to grant him a diploma. Nowadays, more work is in the admission of foreign students to our master program. New rules for guarding the quality of tests and examinations makes the job even heavier, but I have resigned from this office last fall. These problems are for my successor Michael Biehl.

7 Research

It is with research that missing arguments are felt most keenly. Next to the passion for the missing argument there is also the love for the excellent argument. In this respect, the summit of my remembrances is from a lecture by George Kempf in Les Plans sur Baix, Switserland, in 1977. His explanation was rather hazy, so that most of the audience was baffled, but all of a sudden I saw the core of a beautiful argument.

It was as follows. Please, try to imagine a patato. The patato is called *convex* if, for any pair of points of the patato, the whole line segment that connects these points is also in the patato. So, the patato is "filled". It is not a banana or a pear. Of course, Kempf was talking about arbitrary geometric figures, and not specifically about patatoes.

Anyway, try to imagine such a convex patato. For any point P outside the patato, then, there is precisely one point in the patato closest to P. This is because, if there were two points A and B in the patato equally close to P, then the midpoint of the line segment AB is even closer to P.

I do not expect you to be wildly enthusiastic about this. Yet, in the context of Kempf's lecture, it was a beautiful geometric argument with far-reaching consequences.

It is undoubtedly this love for careful arguments that I shared with Edsger Dijkstra and that made it possible to work with him fruitfully. Yet Dijkstra was more finnicky than I am. For him, an argument had to be beautiful, while I am often content with a effective but somewhat ugly argument. For Dijkstra's 60th birthday in 1990, we have written a book under the title "Beauty is our

business" [4]. I have written my personal contribution to this book with special care to comply with Dijkstra's preferences.

7.1 The meaning of programs, angels and demons

After my transition to computer science, my first concern was the question of the meaning of programs. I wanted a mathematical foundation for programming methodology. In programming, it is important to be able to postpone premature design decisions, but this can lead to uncertainty, so called nondeterminism. There are two forms of nondeterminism: angelic nondeterminism in which we assume that the best possible alternative will be chosen, and demonic nondeterminism in which we are prepared for the worst possible scenario. In angelic nondeterminism, we can expect assistence from our personal guardian angel. In demonic nondeterminism, we need to reckon with obstruction by a malicious demon.

I have worked on this mainly between 1986 and 1994. Around 2007, I received a paper from two Englishmen (Morris and Tyrrell) in which angelic and demonic nondeterminism were combined in a way that reminded me of an old attempt of mine to model this, years backward. When I got a second paper about this, some months later, I searched through old manuscripts of mine. Indeed, I found a manuscript with ideas in this direction that ended in failure, but to my surprise I also found a paper from January 1987 in which I succeeded, and obtained results more or less similar to those of Morris and Tyrrell. At that time I was with Dijkstra at Austin, with continuously new ideas around me. This manuscript I had cleanly forgotten.

7.2 The reading group

When Jan van de Snepscheut left for Caltech in 1989, I inherited his Tuesday Afternoon Club. It was a group of staff and students who came together weekly. In Dijkstra's Tuesday Afternoon Club, the idea was to discuss and do research on the spot. This asks for great and continuous creativity from the organizer, something Dijkstra was able to, Jan barely so, but I could not do this. I therefore changed the formula into the carefully reading of scientific papers, mostly from elsewhere, but sometimes our own papers (finished or in the process of being written).

This reading group has existed for 20 years. It has offered me much. We have read papers that inspired me to enter new research areas, and I have subjected many papers to the scrutiny of this group. For the writer of a paper it is very useful, though often frustrating, to observe someone else trying to read and interpret what you have written.

7.3 Concurrency

From 1992 onward, I have moved into the field of the design and correctness of concurrent algorithms. With concurrency, the problem is that several computers, or processes on a single computer, communicate with each other for the sake of some shared goal, or to control a common resource like a data base. This makes programming more difficult because you never know which of them will do the next step. This research area becomes nowadays steadily more important because modern computers are multicore computers. This means that they actually consist of several processors that have to divide their tasks efficiently.

The way to remain certain about aspects of the behaviour of such a distributed system is to analyse which properties *are not made false by any step of the system.* These properties are called the invariants of the system. As the distributed system at any moment can do many different steps, the proof of invariance usually requires many case distinctions.

Humans are not very good in this, because we tend to ignore seemingly innocent cases. I therefore use a theorem prover. This is a computer program with which you can verify mathematical theorems. It has no real intelligence, but it has a good memory. It can easily register and remember what has been proven, and tell you which proof obligations remain.

My shift to concurrency occurred because in the reading group we read a paper [8] of Herlihy, that we could not follow completely. I therefore studied the paper so deeply that, in the end, I came up with a different algorithm for Herlihy's problem. One of the reviewers of that paper raised so many doubts about my argument that I felt forced to study the matter even more deeply. This again led to a proof, but the details were so daunting that whenever I came to the end of the proof, I could not remember whether I had really treated all cases in the beginning.

This led me to use a theorem prover. In Austin, in 1986, I had taken a course of J Moore about his theorem prover NQTHM, but I had never "touched" the prover at the time. This was going to change. I gave myself six months to learn to work with NQTHM and to prove my algorithm. Fortunately, J Moore was always willing to answer my emailed questions about the prover. Thanks to this long-distance coaching, I indeed succeeded in proving the algorithm.

That I like working with a theorem prover, is undoubtedly because it fits with my passion for the missing argument. Between colleagues there is often not enough time to go into the complete verification of a result. There are always other priorities. A theorem prover however is only convinced by the proper argument.

7.4 Jan Friso Groote and Gao Hui

Around 1997, a collaboration with Jan Friso Groote from Eindhoven University emerged, first centered on wait-free memory management [14]. Subsequently, we tried to implement wait-free hashtables. The latter project got stuck when I had proved the first part with the theorem prover NQTHM, and Jan Friso started to prove the second part with his theorem prover PVS, but became educational director.

In the summer of 2000, Gao Hui from China came to Groningen to accompany his wife who had obtained a PhD position in chemistry here. Gao wanted to do a PhD in computer science with me. I had no position for him, but he got a PhD position in statistics. A year later, however, both his supervisors left the institute. Then it was arranged that Gao would shift to computer science and work on a PhD with me.

He worked very hard. After one year, he had seen enough computer science and especially concurrency, that he could try and attack the problem of the wait-free hashtables of Jan Friso and me. Jan Friso came to Groningen for a day to introduce Gao to his PVS proof. It took Gao a year and a half to conclude the proof, and to write a paper about it [5]. Alas, Jan Friso and I had waited too long, and had lost the priority: in the mean time another paper on wait-free hashtables had appeared [17]. Yet, Gao got his doctorate on a beautiful thesis [6] in Groningen, April 2005, with Jan Friso as second promotor and Maurice Herlihy in the reading committee.

We have invited Gao to come to Groningen today, but alas he could not come because of problems with visa and passports.

7.5 Predicting computations

If I am not mistaken, my first encounter with prophecy variables was during the inaugural address [7] of Jan Friso where he conjectured that a correctness proof of Bloom's algorithm would need prophecy variables. This seemed ridiculous to me because I had proved this algorithm already without prophecy variables. With hindsight, however, Jan Friso was probably right, because in my proof I had used a general principle the correctness of which I have proved later using prophecy variables.

What are prophecy variables? They are auxiliary variables that are added to a computer program to predict the future development of the computation. They are useful for the correctness proof of the algorithm, but play no role in the computation itself. They have been introduced in 1991 by Abadi and Lamport [1]. I needed them in 2002 when I wanted to prove something about transactions in data bases. The prophecy variables of Abadi and Lamport, however, were too restricted for my purposes. I had to invent something stronger. I have called these eternity variables.

Reflexion on these issues leads to speculations about predestination, with a supreme being that in one glance can see the complete development of his creation, from the beginning to the end. Here, the development of the creation stands for the execution of the program. I took some inspiration from the books on Dune by Frank Herbert. Actually, it is quite simple: you can view an infinite sequence of numbers as a single object, without taking into account that the numbers are delivered one by one, by a sensor or by throwing dice. The conceptual difficulty is that you imagine to know the final outcome of a process before its conclusion (and without using this knowledge).

7.6 Mutual exclusion

My latest collaboration with someone outside the institute is with Alex Aravind from Northern British Columbia. I have e-mail contact with him since August 2008, but I met him yesterday for the first time, and he is now in the audience.

In 2007, Alex submitted a paper to Information Processing Letters. The paper contained a new interesting algorithm for mutual exclusion, but it was rejected by the journal because it had no convincing proof of correctness. I was one of the anonymous referees, but when the paper was rejected, I contacted the author, because I found it a beautiful algorithm and I could prove its correctness [2].

It is a good collaboration based on complementarity. Alex has very good ideas for algorithms, and I am often able to prove or refute these ideas.

Let me explain his first algorithm as a children party. The children are playing in the garden. When they are thirsty, they can enter the house to get a drink. There is, however, only one glass. The problem is to organize that children never simultaneously grab the glass. This is called *mutual exclusion*. It was proposed by Dijkstra in 1965 for computers (not for children).

Alex Aravind devised the following solution. Suppose there are N children. Make sure that the room with the glass has N corners, numbered from 0 to $N-1$. The glass is in corner 0. Every child in corner 0 can grab the glass. After drinking, the child goes back to play in the garden (or it may go home). When it is thirsty again, it may come back to the room. A child that enters the room counts the number of children in the room. When a child in the room sees that there are not more than k children in the room, it may go to corner k. A child that sees no other children in the room, may therefore immediately go to corner 0 and have a drink. Suppose, however, that three children enter the room at the same time. All three see two other children in the room. They go therefore to corner two, and there is no possibility for progress anymore. Alex solved this by putting a chair in every corner. If a child is in a corner with a number unequal to 0, it may climb onto the chair. When it does so, it pushes the present occupant from the chair. When a child is pushed from the chair in corner k, it may go to corner $k-1$.

This organization indeed guarantees that there is never more than one child

in corner 0. Children can pass one another, but even a clumsy child is passed never more than N times.

I have constructed the proofs of these facts. They are ingeneous, and not of a beauty that Dijkstra would have appreciated. They are effective, however, and verified with the theorem prover PVS.

8 Acknowledgements

Colleagues, staff, and students, of computer science, mathematics, and artificial intelligence in Groningen, I am grateful that I have been able to work with you here for 35 years.

I am grateful to the Groningen University Fund that it has enabled me to do this for 16 years as a professor.

I cannot thank all people with whom I have worked closely, all these years. The list of those people is too long, and if I tried, it would turn out that the list was incomplete.

One exception must be made, however, for Gerard Renardel. Dear Gerard, you came in 1991 as a professor to Groningen to lead the group of fundamental computing science, with me as a member of it. We have collaborated 18 years fruitfully and without conflict. I am grateful to you for this collaboration and for the space you have granted me.

Yet another exception must be made: for my wife Marijke, with whom I am married for 40 years, in three days from now. Dear Marijke, thank you very much. Without you, none of this would have been possible.

9 Finally

Shaking hands with a departing professor is also a mutual exclusion problem. I suggest that we solve this concurrently in a nondeterministic fashion. You need not form a queue, but please take a drink where drink is available, and shake my hand, if it is free and you want to shake it.

References

[1] M. Abadi and L. Lamport. The existence of refinement mappings. *Theor. Comput. Sci.*, 82:253–284, 1991.

[2] A.A. Aravind and W.H. Hesselink. A queue based mutual exclusion algorithm. *Acta Inf.*, 46:73–86, 2009.

[3] A. Doxiadis, C.H. Papadimitriou, A. Papadatos, and A. di Donna. *Logicomix*. Dutch Media Uitgevers, 2009.

[4] W.H.J. Feijen et al., editors. *Beauty is our business, a birthday salute to Edsger W. Dijkstra*. Springer, 1990.

[5] H. Gao, J.F. Groote, and W.H. Hesselink. Lock-free dynamic hash tables with open addressing. *Distr. Comput.*, 17:21–42, 2005.

[6] Hui Gao. *Design and verification of lock-free parallel algorithms*. PhD thesis, University of Groningen, April 2005.

[7] J.F. Groote. We moeten software leren beheersen, 1999. Intreerede, Technische Universiteit Eindhoven.

[8] M. Herlihy. Wait–free synchronization. *ACM Trans. Program. Lang. Syst.*, 13:124–149, 1991.

[9] W.H. Hesselink. Desingularizations of varieties of nullforms. *Inventiones math.*, 55:141–163, 1979.

[10] W.H. Hesselink. A mathematical approach to nondeterminism in data types. *ACM Trans. Program. Lang. Syst.*, 10:87–117, 1988.

[11] W.H. Hesselink. *Programs, Recursion and Unbounded Choice, predicate transformation semantics and transformation rules*. Cambridge University Press, Cambridge, 1992. (Cambridge Tracts in Theoretical Computer Science 27).

[12] W.H. Hesselink. Proof rules for recursive procedures. *Formal Aspects of Comput.*, 5:554–570, 1993.

[13] W.H. Hesselink. Predicate transformers for recursive procedures with local variables. *Formal Aspects of Computing*, 11:616–636, 1999.

[14] W.H. Hesselink and J.F. Groote. Wait-free concurrent memory management by Create, and Read until Deletion (CaRuD). *Distr. Comput.*, 14:31–39, 2001.

[15] D.R. Hofstadter. *Gödel, Escher, Bach: an eternal golden braid*. Vintage Books, 1979.

[16] A. Meijster, J.B.T.M. Roerdink, and W.H. Hesselink. A general algorithm for computing distance transforms in linear time. In J. Goutsias, L. Vincent, and D.S. Bloomberg, editors, *Mathematical morphology and its applications to image and signal processing (Proc. 5th Int. Conf.)*, pages 331–340. Kluwer, 2000.

[17] O. Shalev and N. Shavit. Split-ordered lists: lock-free extensible hash tables. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 102–111. ACM Press, 2003.

[18] A.M. Turing. On computable numbers with an application to the Entscheidungsproblem. *Proc. of the London Mathematical Society 2*, 42:230–265, 1936.

[19] J. van de Snepscheut, editor. *Mathematics of Program Construction*, volume 375 of *LNCS*. Springer V., 1988.