Machines, Languages, and Computation at MIT

Peter J. Denning and Jack B. Dennis

Editor: David Walden

Once upon a time, there was a course at MIT in the Electrical Engineering and Computer Science department called 6.253, Theoretical Models of Computation. It was developed in the time when computer science was not such a large field and all computer scientists were expected to know everything, including theory, programming, architecture, and systems. Rivalries between different parts of the field were just starting to appear.

In the mid-1960s, a tremendous outpouring of new insights from Noam Chomsky and his students in the Department of Linguistics occurred at MIT. They were uncovering unsuspected relationships between formal languages and automata. In particular, they proposed that each category of automata—finite state, pushdown, linear-bounded, and Turing machines—could be characterized by the classes of languages they could recognize. They found that each class of language could be described by the structure of formal grammar needed to define it. They also found that they could construct automata from the grammars and grammars from automata. For example, a language would be classified as pushdown-recognizable if a pushdown automaton could tell whether a given input string is in the language. A context-free grammar could be converted to a description of a pushdown automaton for recognizing a language, and a context-free grammar could be constructed from a pushdown automaton. These insights were extremely valuable in the construction of compilers for higher-level language such as Algol.

Jack Dennis was a faculty member in the Laboratory for Computer Science at the time that Chomsky's students were presenting their results at seminars. He became convinced that it was important for computer scientists to learn this perspective. At that time, the only related books were Marvin Minsky's course notes, which were eventually published by Prentice Hall,[1] and several books on switching theory and logic circuits. These books did not take the view that the automata they discussed were language recognizers. That omission was Jack's motivation for creating the course 6.253. David Kuck and he designed the course and taught it together in the spring of 1965. Peter Denning joined him as a teaching assistant in 1966 and soon became a full partner in refining and extending the notes, developing problem sets, and lecturing.

In 1967, the computer science editor of Prentice Hall, John Davis, found out about the notes for 6.253 and persuaded us (Jack and Peter) to make a book from them. At the time, we envisioned that we could complete the manuscript in 1968. Unfortunately, that did not come to be. Peter was writing his PhD thesis in 1968 and then he joined the electrical engineering faculty at Princeton. Jack found that members of the theory section of the Lab for Computer Science could not accept that a couple of systems guys would teach a good course and write a good book on a formal topic. We were not card-carrying theorists! Because he wanted to devote himself to developing a course on computer systems (numbered 6.232), Jack recruited Joseph Qualitz to take over 6.253 after Peter left. Joe kept the course going and helped complete the book project. The 6.253 course was taught for last time in the fall of 1973.

With all the distractions, and our reduced involvement in teaching the course, it took us 10 years to complete the book. *Machines, Languages, and Computation* (MLC) was published in 1978, a decade later than we had originally estimated.[2]

At Princeton, Peter encountered a similar reaction from the theorists as Jack had at MIT. Princeton had some of computer science's greatest theorists either on the faculty (Jeff Ullman) or closely affiliated (Al Aho and John Hopcroft). Hopcroft and Ullman were working on their own book about automata and languages, which was published in 1979.[3] Peter taught a few sections of the automata theory course from the draft of his MLC book. But the Hopcroft-Ullman book was more popular among the Princeton faculty and students. Jeff and Peter had frequent conversations about the formulation of the MLC book. Jeff liked some parts of MLC, but intensely disliked the construction we had for relating context-free grammars to pushdown automata. He said it was backward from the standard normal forms used in the formal language literature. One day Peter approached Jeff and said that he thought he had a short proof that precedence grammars were deterministic context free. The only existing proof occupied most of Susan Graham's PhD thesis. Peter wanted a short proof that would fit on two or three pages of the MLC book, which was then still in draft. A short proof had eluded Jeff for several years. Peter started presenting draft proofs to Jeff. At first, Jeff kept sending Peter away saying, "There is a bug in your proof." Then one day he said, "I think you've got it. Your unconventional backward construction for pushdown automata gives the key." Peter joined with Aho and Ullman on a paper that described the shorter proof in 1972.[4] The systems guys had found a short proof that eluded the

 Published by the IEEE Computer Society 1058-6180/15/$31.00

theorists, but the theorists thought it was dumb luck for our having got a key definition backward.

While the teaching of the course, we discovered a short proof for the noncomputatiblity of the Correspondence Problem.[5] It was based on a construction where we put the allowable moves of a Turing machine on the dominoes such that you could get the dominoes to spell out a Turing machine computation if and only if the machine halts.[6] We wrote that up and sent it to one of the theory journals. We soon heard from Robert Floyd that he had also found a short proof, which was in press. It did not depend on any kind of domino construction. Thus, we never published our short proof, except of course in our MLC book.[6] Years later, we learned that other teachers were using a domino-type construction to teach the proof to their students,[7] confirming what we already knew—namely, that we had a student-friendly proof even if the theoreticians of the day had shorter mathematical proofs.

The 6.253 course and MLC book brought a systems view to automata and languages. We found that our students struggled with definitions like "A finite state machine is a five-tuple consisting of …" They much preferred when we could draw state graphs and then express the operation of the machine as a program of simple instructions. In our homework exercises, we had students write programs for each of the four categories of machines. Our approach was not as compact and mathematically elegant as the formulation in Hopcroft and Ullman. We tried hard to be student-friendly while bringing the systems view.

A more important difference, we think, is that Hopcroft and Ullman included a chapter on the emerging field of computational complexity. We had briefly considered adding a chapter on complexity but decided against it because we were having enough difficulty reaching closure on the original table of contents, which was defined before the complexity field emerged. In retrospect, that omission was a strategic error for our book.

Despite these difficulties, the book did well and remained in print until 1990. When we pursued our true callings, the modeling and analysis of computer systems, we did much better. The theorists came to us to find out if the assumptions of a current model matched reality and whether we thought a formal result would be of interest to anyone. It turned out well.

References and Notes

1. M. Minsky, *Computation: Finite and Infinite Machines*, Prentice Hall, 1967.
2. P.J. Denning, J.B. Dennis, and J. Qualitz, *Machines, Languages, and Computation*, Prentice Hall, 1978.
3. J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.
4. A. Aho, P.J. Denning, and J.E. Hopcroft, "Weak and Mixed Strategy Precedence Parsing," *J. ACM*, vol. 19, no. 2, 1972, pp. 225–243.
5. The correspondence problem concerns two lists of code words for the same message source and asks whether there is a message that encodes the same in the two codes. For example, with Code1 = {a, ab, bba} and Code2 = (baa, aa, bb}, the message (3,2,3,1) encodes as bbaabbbaa in Code1 and as bbaabbbaa in Code2. Therefore (3,2,3,1) is a solution for these two codes. The general decision problem is for any two codes of the same message source: is there a message encoding the same in both codes?
6. A domino is a block with a string on the top and another string on the bottom. For the correspondence problem, we make a set of domino types with a code word on the top from Code1 and on the bottom from Code2. A solution to the problem exists if we can find a sequence of these dominoes where the string spelled out along the tops matches the sting spelled out along the bottoms. It is possible to make a set of dominoes that spell out Turing machine tapes and control states and represent a Turing machine move from the top to bottom string on a domino. The tops and bottoms match if and only if the Turing machine represented halts. Thus, a solution to the correspondence problem would solve the halting problem. The Wikipedia article, "Post correspondence problem," refers to a 2005 book chapter by Michael Sipser, "A Simple Undecidable Problem," *Introduction to the Theory of Computation*, 2nd ed., Thomson Course Technology, 2005, pp. 199–205. Our MLC book contains a similar proof discovered over three decades earlier.
7. Sipser, "A Simple Undecidable Problem," *Introduction to the Theory of Computation*, 2nd ed., pp. 199–205.

Peter J. Denning *distinguished professor, chair of the Computer Science Department, and director of the Cebrowski Institute for Innovation and Information Superiority at the Naval Postgraduate School. Contact him at pjd@denningassociates.com.*

Jack B. Dennis *is a professor emeritus of Computer Science and Engineering at MIT. Contact him at dennis@csail.mit.edu.*