Profile | DOI:10.1145/3460734

Neil Savage

Getting Down to Basics

2020 ACM A.M. Turing Award recipients Alfred Aho and Jeffrey Ullman helped develop formal language theory, invented efficient algorithms to drive the tasks of a compiler, and put them all together in 'The Dragon Book.'

WRITING THE CODE to make a computer perform a particular job could be a Herculean task, back in the 1950s and 60s.

"In the early 1950s, people did numerical computation by writing assembly language programs," says Alfred V. Aho, professor emeritus of computer science at Columbia University. "Assembly language is a language very close to the operations of a computer, and it's a deadly way to program. It's slow, tedious, and expensive."

Of course, people can program at higher levels of abstraction, but that requires translating the higher-level language into a more basic set of instructions the machine can understand. Compilers that efficiently perform that translation exist nowadays in large part due to the work of Aho and Jeffrey D. Ullman, professor emeritus of computer science at Stanford University. Their contribution to both the theory and practice of computer languages has earned them the 2020 ACM A.M. Turing Award.

"Compilers are responsible for generating the software that the world uses today, these trillion lines of software," Aho says. "They've been around for a long time, and the modern world wouldn't exist without programming language translators."

The two men helped develop formal language theory. They invented efficient algorithms for performing lexical analysis, syntax analysis, code generation, and code optimization, all essential tasks for a compiler. And they put that all together into *Compilers: Principles, Techniques, and Tools*, the definitive textbook on the subject, now known simply as "The Dragon Book" for its colorful cover illustration.

"No matter what you do in life, you need to understand how to use computers, which usually means being able to write at least simple code in some appropriate language."

The two met in 1963 while waiting in line to register for the Ph.D. program at Princeton University. Ullman, now 78, received his B.S. from Columbia that year, and Aho, 79, had just graduated from the University of Toronto with a B.S. in engineering physics. After each received his doctorate from Princeton a few months apart, in 1966 and 1967, they both joined Bell Laboratories, a hotbed for science and innovation.

"Academia has sort of always been my game plan," Ullman says, adding that he didn't feel assistant professors were particularly well treated at the time. "Today, assistant professors are really treated well in academia, especially in the field of computer science, where we recognize it's often the young people who have the greatest ideas," he says. When he was offered a job at Columbia he jumped at the chance, but kept traveling to Bell Labs about once a week to continue collaborating with Aho, until he moved to Stanford University in 1979. Aho, meanwhile, stayed at Bell for nearly 30 years, until he joined the Columbia faculty in 1995.

The researchers laid out a generic framework that provided the roadmap for how a compiler works. First comes

PHOTO BY ALEXANDER BERG

Alfred Aho and Jeffrey Ullman in the corridor of their former office spaces at Bell Labs.

lexical analysis, in which the compiler scans the program to identify tokens, which are analogous to the individual words in a sentence. Next comes a parser, which analyzes the syntax of the program, and checks whether the tokens fit together. "Not every sequence of tokens is a well-constructed program," Aho says. Next comes semantic analysis, which checks whether the sequences make sense as a whole.

The two shrunk the size and increased the speed of a left-right parser, which aims to read the program in sequence from left to right without doubling back, but which is allowed to peek ahead to determine meaning. For instance, if the program wants to perform the operation A+BxC, the parser can see the A+ and know it hasn't reached the end of that particular element, so it keeps going. It eventually sees that it has to multiply B and C first, then add A, just because the creators of the compiler defined that order of precedent for arithmetic operations.

The trio of lexical, syntax, and semantic analysis makes up the front end of the compiler, which produces an intermediate representation of the program, which can then be optimized. If it turns out, for instance, that C equals 1, so that BxC=B, the program can be simplified by changing it to merely A+B. "That is a kind of optimization that can be performed to make the ultimate target program run faster," Aho says. The last step is to map the optimized intermediate to an assembly language, which can translate it directly into machine language for the particular type of computer.

Enter the Dragon

Aho and Ullman spelled all this out in the Dragon Book, which has since had two more editions and added Ravi Sethi, and later Monica Lam, as coauthors. It was Ullman's idea to depict a knight battling a dragon on the cover. The dragon represents "complexity of compiler design," Ullman explains, and "many of the weapons of the knight are the tools that we advocated people use," such as data flow analysis and syntax-directed translation. The first version showed a green dragon, the second was red, and the third, drawn by Ullman's son Scott, is purple.

The dragon was red when Krysta Svore, now head of the quantum computing group at Microsoft Research, was a student in Aho's class at Columbia. Even as advanced an application as a quantum program relies on the same ideas the two developed in the 1960s and 1970s. "What they're winning for is part of the foundation of what I and my team are working on for this next type of computer," she says. "What they developed is still critical for writing programs across smartphones, across quantum computers, across your laptop. These technologies are very much forever critical."

"What they developed is still critical for writing programs across smartphones, across quantum computers, across your laptop. These technologies are very much forever critical."

In Aho's class, he asked students to write their own domain-specific language. Svore wrote a quantum computer language, and enjoyed the experience so much that she asked Aho to become one of her thesis advisers.

The men streamlined the process of creating compilers even further by developing tools such as the Lex lexical analyzer generator, and the yacc parser generator, both of which automate the building of compiler components. Aho, along with Peter Weinberger and Brian Kernighan, also developed AWK, a "little language" to simplify some common data processing tasks. "To do these operations in C would require you to write 100 or 200 lines of C," Aho says. "We could specify these routine data processing tasks with one or two lines of AWK."

Aho and Ullman each have written several other books, both together and with other authors, including *The Design and Analysis of Computing Algorithms* with previous Turing Award recipient John Hopcroft, which laid down a framework for analyzing and teaching algorithms. They've received other awards for their work, including IEEE's John von Neumann Medal in 2010 for outstanding contributions to computer science.

The Turing Award comes with a $1-million cash prize that Aho and Ullman will split. Neither is sure yet what he'll do with the money. After a year of laying low due to the coronavirus pandemic, Ullman says, "I wouldn't mind going out to a restaurant and having a nice meal."

Broader Applications

The men say young people studying computer science, and even those who are not, could benefit from considering how thinking algorithmically fits into the wider world. "The future of computer science lies in applying the ideas more broadly," Aho says. "They should study not just computer science, but another discipline like biology, because there are fascinating opportunities for the application of computer science ideas to biological problems like how does the body work, how does the disease work?" The most intriguing question computer scientists might address, he suggests, is "how does the brain work?"

Ullman says everybody should be able to think about the world computationally, no matter what they do, just as people who are not professional writers ought to know how to write. Similarly, he says, everyone should be able to describe things precisely, even at such a high level of abstraction that it may not look like programming. "You've got to say what you mean and mean what you say," he says. "No matter what you do in life, you need to understand how to use computers, which usually means being able to write at least simple code in some appropriate language."

Neil Savage is a science and technology writer based in Lowell, MA, USA.

Watch the recipients discuss their work in the exclusive *Communications* video. https://cacm.acm.org/videos/2020-acm-turing-award