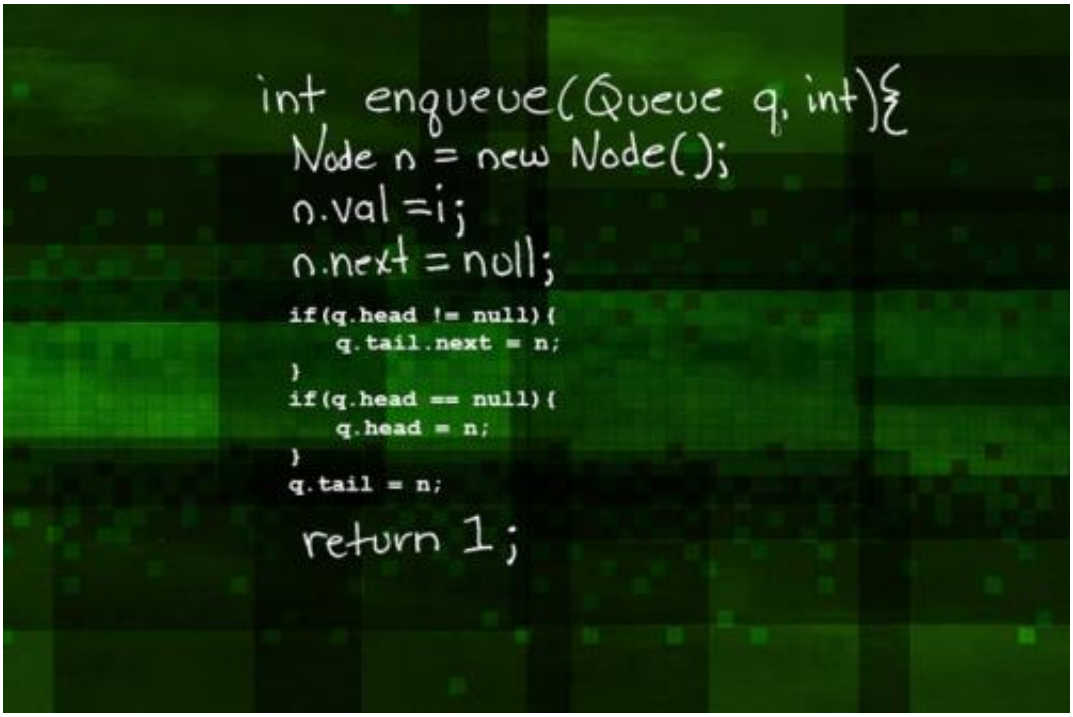
System that automatically fills the gaps in programmers' code improved

February 25 2014, by Larry Hardesty



Credit: Christine Daniloff/MIT

Since he was a graduate student, Armando Solar-Lezama, an associate professor in MIT's Department of Electrical Engineering and Computer Science, has been working on a programming language called Sketch, which allows programmers to simply omit some of the computational details of their code. Sketch then automatically fills in the gaps.

If it's fleshed out and made more user-friendly, Sketch could ultimately make life easier for software developers. But in the meantime, it's proving its worth as the basis for other tools that exploit the mechanics of "program synthesis," or automatic program generation. Recent projects at MIT's Computer Science and Artificial Intelligence Laboratory that have built on Sketch include a system for automatically grading programming assignments for computer science classes, a system that converts hand-drawn diagrams into code, and a system that produces SQL database queries from code written in Java.

At this year's Verification, Model Checking, and Abstract Interpretation Conference, Solar-Lezama and a group of his students—grad students Rohit Singh, Rishabh Singh, and Zhilei Zu, along with MIT senior Rebecca Krosnick—described a new elaboration on Sketch that, in many cases, enables it to handle complex synthesis tasks much more efficiently. The researchers tested the new version of Sketch on several existing applications, including the automated grading system. In cases where the previous version would "time out," or take so long to reach a solution that it simply gave up, the new version was able to correct students' code in milliseconds.

Sketch treats program synthesis as a search problem. The idea is to evaluate a huge range of possible variations on the same basic program and find one that meets criteria specified by the programmer. If the program being evaluated is too complex, the search space balloons to a prohibitively large size. In their new paper, the researchers find a way to shrink that search space.

Chain of command

"When you're trying to synthesize a larger piece of code, you're relying on other functions, other subparts of the code," Rishabh Singh explains. "If it just so happens that your system only depends on certain properties

of the subparts, you should be able to express that somehow in a high-level language. Once you are able to specify that only certain properties are required, then you are able to successfully synthesize the larger code."

For instance, Singh explains, suppose that one of the subparts of the code is a routine for finding the square root of a number, and a higher-level function relies on the results of that computation. If the previous version of Sketch were trying to evaluate variations of the high-level function, for each variation, it would also have to evaluate variations of the square-root function. Since finding square roots is a complex process, that would make the search prohibitively time-consuming.

With the new version of Sketch, however, the programmer can simply specify conditions that the square-root function has to meet: The output multiplied by itself must equal the input. Now, Sketch can satisfy itself that the square-root function it comes up with meets that criterion and move on to the higher-level function. It doesn't need to re-evaluate the square-root function at every pass.

In fact, this places a slightly greater onus on the programmer, who now has to reason about the criteria that each low-level function must meet. But it allows Sketch to handle much more complicated problems.

Immediate prospects

Solar-Lezama concedes that it will take a good deal of work before Sketch is useful to commercial software developers. "The application as a tool-building infrastructure, using it to build higher-level systems on top of it, we've demonstrated very convincingly by building a variety of systems that do things that couldn't be done before," he says.

He has, however, conducted usability studies with Sketch, recruiting

MIT undergraduates with only a semester's worth of programming experience to test it. In all cases, he says, the students successfully used Sketch to produce working code. But in many cases, the missing code took an unacceptably long time to synthesize, because of the way the students had described the problem.

"It still requires a level of expertise and understanding about the underlying technology in order for it not to blow up," Solar-Lezama says. "As far as the more ambitious goal of everybody dumping C and using Sketch instead, we'd still have to push quite a bit."

As Rajeev Alur, a professor in the Department of Computer and Information Science at the University of Pennsylvania, explains, the new paper draws on principles from the field of "formal verification," which, Alur says, investigates methods for "checking the correctness of programs using automated reasoning."

"In verification, people have always used modular reasoning as a technique to make it scale to more interesting systems," Alur says. "What this paper does is take some of those ideas and meshes them nicely with the synthesis routines they have in Sketch."

Alur acknowledges that "having a general software developer use [Sketch], maybe that's not realistic in the foreseeable time." But, he says, "even now it could be used in very specific, specialized tasks. If you're trying to optimize some piece of code for some reason, instead of doing all that fine-tuning of the code manually, now a system like Sketch could do it."

Provided by Massachusetts Institute of Technology