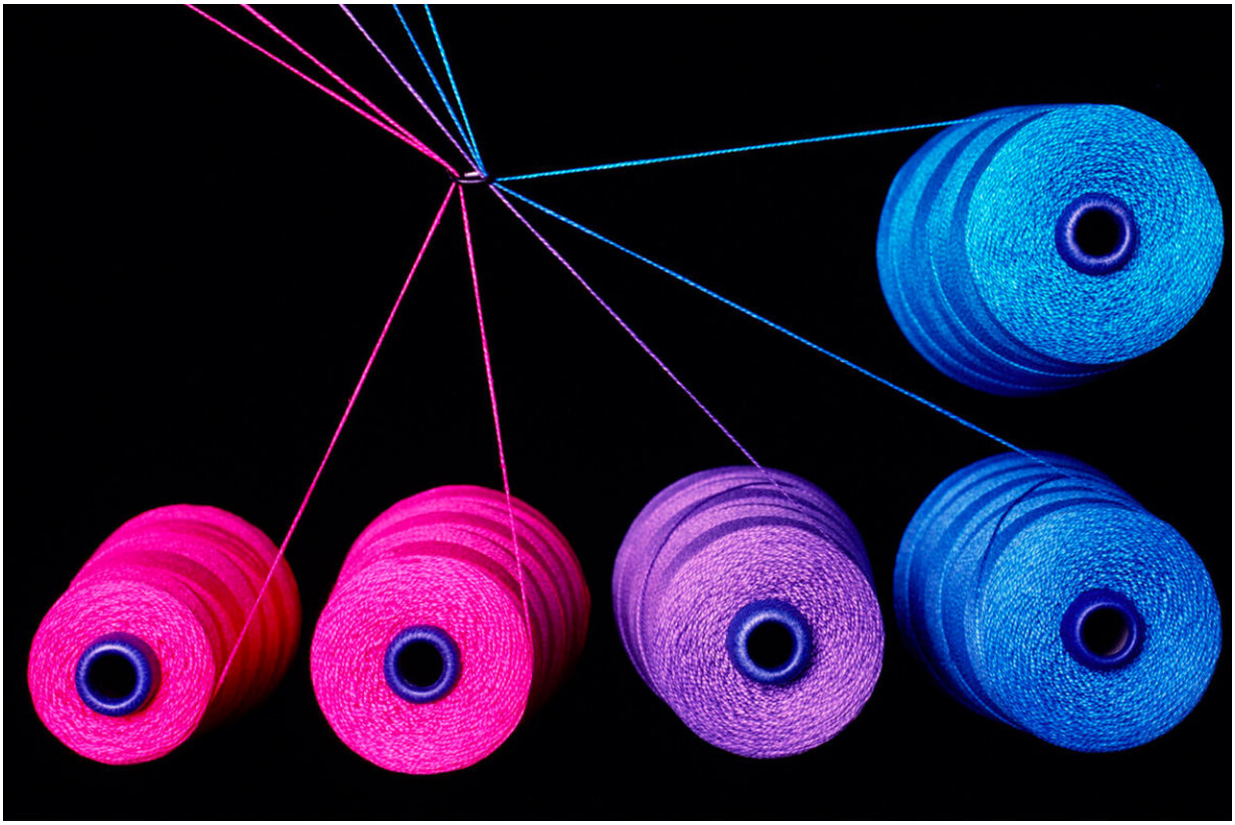
New method ensures complex programs are bug-free without testing

June 16 2020



Multithreading is one common form of concurrent execution, allowing different instructions in a program to be processed simultaneously by one or more CPU cores. Credit: University of Michigan

A team of researchers have devised a way to verify that a class of complex programs is bug-free without the need for traditional software

testing. Called Armada, the system makes use of a technique called formal verification to prove whether a piece of software will output what it's supposed to. It targets software that runs using concurrent execution, a widespread method for boosting performance, which has long been a particularly challenging feature to apply this technique to.

The collaborative effort between the University of Michigan, Microsoft Research, and Carnegie Mellon was recognized at ACM's Programming Language Design and Implementation (PDLI 2020) with a Distinguished Paper Award.

Concurrent programs are known for their complexity, but have been a vital tool for increasing performance after the raw speed of processors began to plateau. Through a variety of different methods, the technique boils down to running multiple instructions in a program simultaneously. A common example of this is making use of multiple cores of a CPU at once.

Formal verification, on the other hand, is a means to demonstrate that a program will always output correct values without having to test it with a full range of possible inputs. By reasoning about the program as a mathematical proof, programmers can demonstrate that bugs or errors are impossible and that its execution is airtight. This overcomes the shortcoming common to all programs, even without concurrency, that testing something exhaustively can be either impractical or actually impossible.

"Fundamentally, unless you try all the possible inputs, you may miss something," says Prof. Manos Kapritsos, co-author on the paper. "And in practice, people do miss things. The systems we're talking about are very complex, there's no way that they can exhaustively try all the behaviors of the system."

Formal verification offers an alternative to this need for exhaustive testing. But the process of generating a satisfactory proof turns out to be very difficult and time-consuming, especially as you delve into programs with the added complexity of concurrency.

"The main challenge in concurrent programs comes from the need to coordinate many different threads of code together," says Upamanyu Sharma, co-author who worked on the project as an undergraduate at U-M. "To verify that multi-threaded programs are correct, we have to reason about the huge number of interleavings that are possible when multiple methods run at the same time."

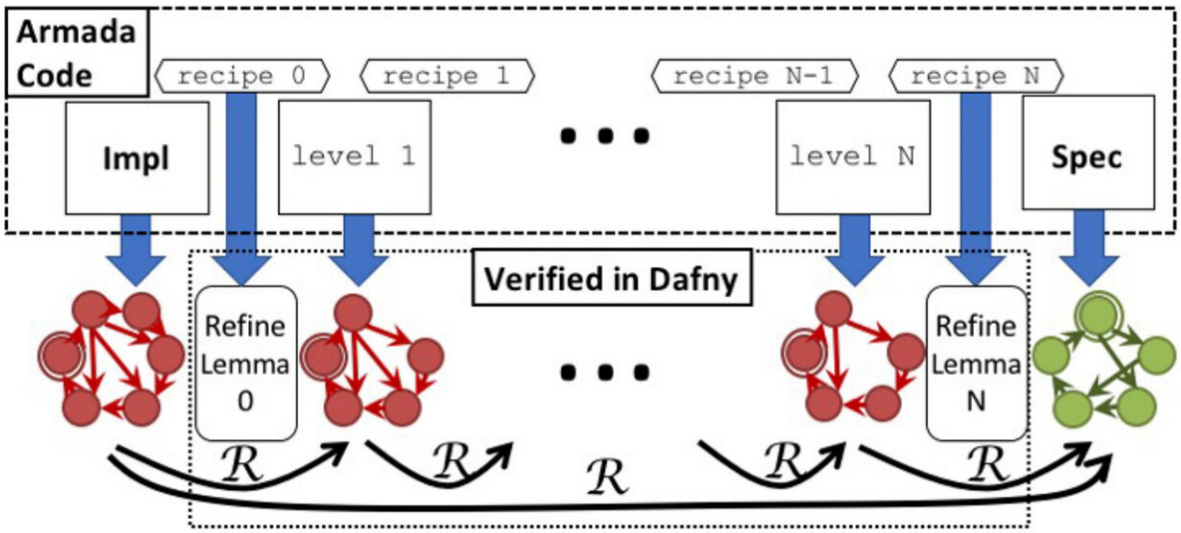
This huge number of branching possibilities is difficult to conceptualize and express through logical formulas.

To date, a variety of proof methods have been designed to deal with different types of concurrency. In this project, the researchers set out to design a single framework that allows a user to apply many of these techniques to verify a single program, with the ultimate goal of cutting down the effort up front as much as possible.

Armada works by passing a system designed with concurrency through a series of transformations until it's broken down into a much simpler representation. The developer just has to prove that each simplified step really is representative of the more complex program from the previous step. To do this, the developer uses Armada's high-level syntax to describe the simpler program and indicate one of the proof methods needed to support the transformation.

"After every transformation, you want to reason that the system maintains its correctness or is equivalent to the previous one," Kapritsos explains.

The proof itself is then generated automatically for each step by Armada and run through a prover for verification. In the event the proof fails, the user changes their description or proof method and generates a new one.



Armada allows a developer to write a short description of their software and the proof methods they want to use and then generate a significantly longer full proof automatically. Credit: University of Michigan

In the end, the developer has a simple, high-level specification for the entire system. They haven't made any changes to the system itself, just reasoned about its functionality in increasingly abstract steps that are each still representative of the functioning of the whole program.

"Part of the goal is to support high performance," says Kapritsos. "We don't want you to rewrite your system just so it can be verifiable."

In the world of verifying concurrent programs, this is perhaps the most

low-effort technique available. In demonstrations, the team used Armada to verify four concurrent case studies and show that it achieves performance equivalent to that of unverified code.

In one test, the team used Armada to verify a data structure implementation Kapritsos deems "notoriously difficult to reason about." Called a lock-free queue, the structure is a standard queue (in which the first data stored is the first to be removed) without the locking mechanism typically needed to ensure that only one thread can access a resource at a time in a concurrent program. This implementation provides better concurrent performance over a locking queue, akin to using a roundabout instead of a stoplight for traffic control.

Removing that lock introduces a lot of subtlety to the algorithm, and typically requires long, tedious case analyses to prove its correctness. Using Armada, the team generated such a proof that ended up 24,540 lines of code long—while only writing 70 lines of code themselves.

Armada uses a verifier to determine that any proof methods in its library are sound, and it can be extended with more proof methods in the future.

The authors hope that this shorter pipeline will encourage the broader use of formal verification outside of the most critical systems where the technique is already justified.

"The aim of our work was to show that it is possible to verify high-performance concurrent code with little effort," says Sharma. "While successful in this endeavor, 'low-effort' is relative, and verification is already associated with a great deal of time and energy. For less critical applications, program testing and some static analysis is usually deemed sufficient, especially given its comparatively low cost, even though they do not provide strong guarantees of correctness."

"This is not the end, you can always reduce the effort even more," Kapritsos says, "but we're trying to leverage as much automation as possible, making it as simple as we can for the programmer to take any of these steps."

Broader adoption will rely on continued efforts to lower the cost of formally verifying real systems, he says, one day finally tipping the balance against easier, but incomplete, techniques.

"Some people say that verification of concurrent programs lags behind non-concurrent programs by a decade," Sharma says, "and there is plenty of work to be done."

This system was presented in the paper "Armada: Low-Effort Verification of High-Performance Concurrent Programs."

More information: Jacob R. Lorch et al. Armada: low-effort verification of high-performance concurrent programs, *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation* (2020). DOI: 10.1145/3385412.3385971

Provided by University of Michigan