Research Statement

Ben Little

Sep 13, 2023

The solutions for human problems will come from human minds. The more minds that are empowered to articulate these colutions, the more opportunities we will have for solving our most challening problems. Compters are a powerful way for human minds to articulate their solutions, and programming languages form the basis how humans interact with computers.

However current trends in programming language design make it difficult for most people to engage meaningfully with computers. Today, learning to code is an esoteric practice that involves memorizing a vast array of approximate methods with little insight into what we are actually instructing the computer to do. These approximations are not compositional–a collection of mostly-correct systems together form a mostly-incorrect system. As a result, computer systems fail to scale, becoming brittle as the number of edge cases grows exponentially with the number of interconnected components. Formal verifiction, or the process of proving the correctness of a system, is the only path forward for building truly scalable interconncted systems.

So the apparent solution is to teach people to write proofs of correctness for their software and hardware. Yet a barrier stands in the way of this approach: Most programming languages capable of expressing proofs were not built to be accessible to a wide audience. They are esoteric, hard to learn, and while techincally correct, they do not make this correctness obvious. As a result they remain almost exclusively in the toolkit of researchers, academics, and few brave but highly skilled software engineers.

One method that has been successful for increasing the number of people who are able to write computer programs is the design of Visual Programming Languages (VPLs). VPLs describe computational processes as flow chart, or directed graph, of smaller processes. These processes can then be composed into yet larger processes, and so on. VPLs like National Instruments' LabView have enabled domain experts and engineers to describe computational processes without ever needing to learn a language-based programming paradigm like C, Python, or Haskell. However existing VPLs suffer from a lack of expressivity needed to prove the correctness of the computations they describe.

My research is an effort to bridge this gap: To provide a VPL that is both

easy to use and fully expressive in terms of proving the correctness of programs. Technically, this involves designing a VPL capable of modeling dependent type theory. The theoretical basis of such a VPL is that of string diagrams for cartesian closed categories. By refining existing work on such string diagrams into a diagrammatic calculus, it should be possible to construct a VPL meeting the above requirements.

My hope is that this effort will move us closer to a world where the end-users of computer systems can have confidence and understanding, as opposed to mere trust, that these systems are performing tasks exactly as they intended. With more visibility into the genuine inner workings of software and hardware, we will gain an opportunity to engage with computers in a way that is wise. We may be less likely to fool ourselves into thinking that an unsafe system is safe, or that we have solved a problem that is in fact unsolvable.