Abstract

Programming has become an important skill in today's technology-driven world. It is a complex activity because of which programmers make mistakes in their software. Student programmers make mistakes in their programs due to lack of programming experience and also due to lack of understanding of the algorithmic concepts being used. An experienced programmer in the industry, on the other hand, makes mistakes in the software he develops because of the mere complexity of an industry-grade software. Debugging one's programs, i.e., finding and fixing faults is one of the challenging activities for a programmer. This necessitates the need to develop automated techniques that are effective in assisting programmers repair faults in their programs.

In this thesis, we present new computer-assisted approaches to help programmers repair their programs. Finding repair for a fault in the program is essentially a search over the space of all possible repairs. Traditional repair techniques attempt to output a complete repaired program that eliminates the fault in the original program. This ambitious goal of finding a complete repair is often unachievable because the search space becomes too large to navigate efficiently. The key insight of our work is that programmers only need guidelines that help them understand the faults they made and suggestions on how to fix them, as opposed to a complete repair. Therefore, this thesis proposes a novel perspective of solving the program repair problem, where we generate textual hints that direct the programmer on how to fix the fault. A hint specifies which part of the program needs modification as well as suggests how to modify it. The programmer uses them as guidance to arrive at a complete repair. For student programmers, our hints help them reflect on the mistakes they made and improve their understanding of the problem being solved in the program. For experienced programmers, our

hints help them easily identify the cause of fault and guide them to a repair. Our approaches use novel combinations of syntactic, symbolic and statistical reasoning techniques to achieve this goal of semi-automated program repair.

Our first contribution is an algorithmic technique for automated synthesis of repair hints. Our technique takes as input a faulty program and a set of test cases and outputs a ranked list of repair hints that suggest how to rectify a faulty statement in the program. In this technique, we leverage symbolic execution and statistical correlation analysis to identify expressions that are likely to occur in the repaired code. Then we use syntactic pattern matching to combine these expressions algorithmically and generate repair hints. Since we restrict ourselves to finding "building blocks" of repair, our hints are very effective in guiding the programmer to the final repair, even in scenarios where a complete repair cannot be obtained. We implemented this technique to develop a tool called "MintHint", that performs automated synthesis of repair hints for C programs. We evaluated the effectiveness of MintHint by performing a user study and found that programmers who used MintHint were able to repair more faults compared to those who didn't. In addition, they obtained the repair 5.8 times faster.

Our next contribution is a semi-supervised feedback generation methodology for student programming assignments. In this methodology, we take a set of student submissions as input and generate personalized, verified feedback for each submission that suggests modifications to fix faults in the program, if any. Student submissions are first clustered by solution strategy, which is followed by an automated feedback generation phase. We use unsupervised clustering to reduce burden on the instructor in providing reference implementations for each possible solution strategy. The instructor is called upon simply to identify a correct submission in each cluster, which acts as the reference implementation for all submissions in the cluster. For the next phase, we propose a novel counter-example guided feedback generation algorithm that compares a student submission with a reference implementation to generate verified feedback. We implemented this methodology in a tool "CoderAssist" and evaluated it on programs implementing iterative dynamic programming algorithms. We collected a dataset of 2226 student submissions to 4 programming problems from the programming contest site CodeChef and successfully generated verified feedback for 85% of them.