Using a probabilistic model to predict bug fixes

Mauricio Soto
Carnegie Mellon University
Pittsburgh PA
mauriciosoto@cmu.edu

Claire Le Goues
Carnegie Mellon University
Pittsburgh PA
clegoues@cs.cmu.edu

***Abstract*—The abstract goes here.**

I. Introduction

Automatic program repair has attracted a lot of attention in the last several years due to its ability to tackle complex and expensive tasks such as being able to localize errors and fixing code with minimal interaction with human programmers. One of the most well known approaches of automatic program repair is the generation and validation approach. In this approach we start with source code that has one or several bugs on it; and we have a test suite which has passing test cases and failing test cases. The passing test cases confirm the expected behavior of the program, and the failing test cases expose the deviation of the expected behavior with the actual behavior.

These approaches have a wide variety of mutation operations, which are small changes that can be applied to the source code in order to modify its behavior. In order to pick which mutation operator will be used in a specific location, we first pick a fault location, which is a point in the code where we think the fault is located. This can be done in various ways and there is a whole set of possible approaches that can tell you with different levels of certainty what location to pick. This is not the step that we will be analyzing in this paper. We will be analyzing the next step: After we have a fault location, then it goes through a process in which it is analyzed which mutation operators can be applied to that specific location, or if the context of that location doesn't allow some mutation operators to be used in a particular location.

Once this process is finished, we have a set of possible mutation operators that can be applied to a location. At this point, it is chosen at random which mutation operator will be used from the set of mutation operators that can be applied at this location.

In order to pick the mutation operator, the probabilities of picking each of them are equally distributed; this means that from the set of mutations that can be applied to this location, all of them have the same probability of being chosen. Where, in reality, some mutation operators are much more common than others and the fact that we are choosing from them equally is making us often pick mutation operators that are less likely to actually fix the bug we are trying to fix.

In order to tackle this problem we have developed a probabilistic model which entails different probabilities for each of the mutation operators based on empirical data that describes how to human programmers fix their code.

With the probabilistic model we are able to chose from the set of possible mutation operators that can be applied to a particular location using the probabilities which describe how often do each of the mutation operators is able to fix bugs when used in real projects. If human programmers are more likely to use a particular mutation operator, then that mutation operator will have a higher probability of being picked than other mutation operators.

II. Background

One of the most well known approaches to automatic program repair is the generate and validate approach which is detailed in Figure 1. In this approach we start with source code that has one or various bugs to be fixed, and a test suite which contains passing test cases to assure the expected behavior of the program, and failing test cases which expose the behavior of the program that is expected but the output of the expected behavior does not match the output of the source code as is.

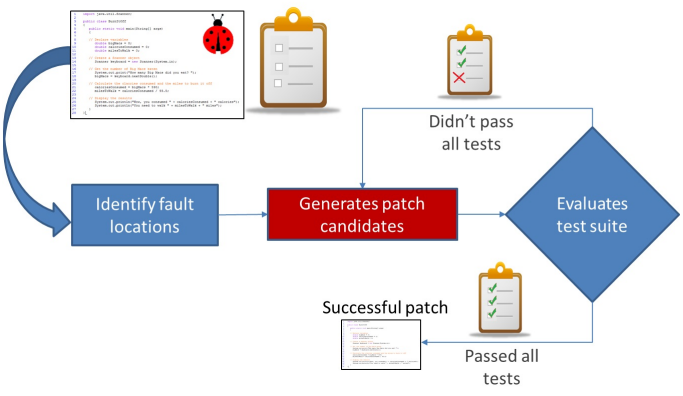

Fig. 1. Generate and validate approach

The first step of this process is to localize the error to a particular statement. There is an extense list of possible approaches to take in order to perform this tasks **List several fault loc papers**, but this paper will not focus on this.

A. General mutations

Subsection text here.

B. Template based mutations

Subsection text here.

III. Build the model

Section text here.

IV. Sanity Check

Section text here.

A. 10 fold cross validation

Subsection text here.

B. Small Example

Subsection text here.

V. Evaluation

Section text here.

A. Single line bugs

Subsection text here.

B. Multi line bugs

Subsection text here.

VI. Discussion

Section text here.

VII. Conclusion

The conclusion goes here.

Acknowledgment

The authors would like to thank...

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.