# Overfitting in Semantics-based Automated Program Repair

Xuan-Bach D. Le
Singapore Management University
dxb.le.2013@phdis.smu.edu.sg

Ferdian Thung
Singapore Management University
ferdiant.2013@phdis.smu.edu.sg

David Lo
Singapore Management University
davidlo@smu.edu.sg

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

## ABSTRACT

Existing APR techniques can be generally divided into two families: semantics- vs. heuristics-based. Semantics-based APR uses symbolic execution and test suites to extract semantic constraints, and uses program synthesis to synthesize repairs that satisfy the extracted constraints. Heuristic-based APR generates large populations of repair candidates via source manipulation, and searches for the best among them. Both families largely rely on a primary assumption that a program is correctly patched if the generated patch leads the program to pass all provided test cases. Patch correctness is thus an especially pressing concern. A repair technique may generate overfitting patches, which lead a program to pass all existing test cases, but fails to generalize beyond them. In this work, we revisit the overfitting problem with a focus on semantics-based APR techniques, complementing previous studies of the overfitting problem in heuristics-based APR. We perform our study using IntroClass and Codeflaws benchmarks, two datasets well-suited for assessing repair quality, to systematically characterize and understand the nature of overfitting in semantics-based APR. We find that similar to heuristics-based APR, overfitting also occurs in semantics-based APR in various different ways.

## KEYWORDS

Automated Program Repair, Program Synthesis, Symbolic Execution, Patch Overfitting

**ACM Reference Format:**
Xuan-Bach D. Le, Ferdian Thung, David Lo, and Claire Le Goues. 2018. Overfitting in Semantics-based Automated Program Repair. In *ICSE '18: ICSE '18: 40th International Conference on Software Engineering , May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 1 page. https://doi.org/10.1145/3180155.3182536

## 1 INTRODUCTION

In this article, we comprehensively study overfitting in semantics-based APR. We perform our study on recent state-of-the-art semantics based APR tools [1, 2, 4]. We evaluate the techniques on a subset of the IntroClass [3] and Codeflaws benchmarks [5], two datasets well-suited for assessing repair quality in APR research. Both consist of many small defective programs, each of which is associated with two independent test suites. One test suite can be used to guide the repair, and the other is used to assess the degree to which the produced repair generalizes. This allows for controlled experimentation relating various test suite and program properties to repairability and generated patch question.

Overall, we show that overfitting does indeed occur with semantics based techniques. We characterize the relationship between various factors of interest, such as test suite coverage and provenance, and resulting patch quality. We observe certain relationships that appear consistent with results observed for heuristic techniques, as well as results that stand counter to those achieved on them. These results complement the existing literature on overfitting in heuristic APR, completing the picture on overfitting in APR in general. This is especially important to help future researchers of semantics-based APR to overcome the limitations of test suite guidance. We argue especially (with evidence) that semantics-based program repair should seek stronger or alternative program synthesis techniques to help mitigate overfitting.

Our contributions are as follows:

(1) The first study to show that semantics-based APR can produce patches that overfit.
(2) We find that, in some cases, results are interestingly inconsistent with that of heuristic approaches.
(3) We substantiate that using multiple synthesis engines could be one possible approach to help semantics-based APR generate correct patches for a larger number of bugs.
(4) We present implications and observations for how to improve semantics APR. For example, one possible improvement is to generate many patches and rank them based on potential likelihood of correctness.

## REFERENCES

[1] Xuan-Bach D. Le, Duc-Hiep Chu, David Lo, Claire Le Goues, and Willem Visser. 2017. JFIX: semantics-based repair of Java programs via symbolic PathFinder. In *ISSTA*. 376–379.
[2] Xuan-Bach D. Le, David Lo, and Claire Le Goues. 2016. Empirical Study on Synthesis Engines for Semantics-Based Program Repair. In *ICSME*. 423–427.
[3] Claire Le Goues, Neal Holtschulte, Edward K. Smith, Yuriy Brun, Premkumar T. Devanbu, Stephanie Forrest, and Westley Weimer. 2015. The ManyBugs and IntroClass Benchmarks for Automated Repair of C Programs. *IEEE Trans. Software Eng.* (2015), 1236–1256.
[4] Sergey Mechtaev, Jooyong Yi, and Abhik Roychoudhury. 2016. Angelix: scalable multiline program patch synthesis via symbolic analysis. In *ICSE*. 691–701.
[5] Shin Hwei Tan, Jooyong Yi, Yulis, Sergey Mechtaev, and Abhik Roychoudhury. 2017. Codeflaws: a programming competition benchmark for evaluating automated program repair tools. In *ICSE*. 180–182.