QuixBugs: A Multi-Lingual Program Repair Benchmark Set Based on the Quixey Challenge

Derrick Lin*
Independent Researcher, USA
drrckln@gmail.com

James Koppel
MIT
Cambridge, Massachusetts, USA
jkoppel@mit.edu

Angela Chen
Google, Inc
Mountain View, California, USA
angchen@google.com

Armando Solar-Lezama
MIT
Cambridge, Massachusetts, USA
asolar@csail.mit.edu

Abstract

Recent years have seen an explosion of work in automated program repair. While previous work has focused exclusively on tools for single languages, recent work in multi-language transformation has opened the door for multi-language program repair tools. Evaluating the performance of such a tool requires having a benchmark set of similar buggy programs in different languages. We present QuixBugs, consisting of 40 programs translated to both Python and Java, each with a bug on a single line. The QuixBugs benchmark suite is based on problems from the Quixey Challenge, where programmers were given a short buggy program and 1 minute to fix the bug.

CCS Concepts • **Software and its engineering → Software maintenance tools**;

Keywords automated program repair; benchmark

ACM Reference Format:
Derrick Lin, James Koppel, Angela Chen, and Armando Solar-Lezama. 2017. QuixBugs: A Multi-Lingual Program Repair Benchmark Set Based on the Quixey Challenge. In *Proceedings of 2017 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity (SPLASH Companion'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3135932.3135941

*This paper reflects work completed while Derrick was an intern at Tarski Technologies, a startup founded by James Koppel which specialized in automated program repair

SPLASH Companion'17, October 22–27, 2017, Vancouver, Canada
© 2017 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-5514-8/17/10.
https://doi.org/10.1145/3135932.3135941

1 Introduction

This need to reimplement program repair tools for other languages motivated the development of ASTOR [7], and prompted the authors of GenProg, originally for C, to reimplement their tool from scratch for Java [4].

Recent work [2] has opened the doors for *multi-language transformation*, where a single program transformation tool can be specialized to work on many different languages, requiring only slightly more work than building a transformation for one language. As program repair is itself a form of source-to-source transformation, and program repair algorithms are often based on program-transformation techniques [1, 5, 6], this opens the possibility for creating multi-language program repair tools. Such a development would be quite significant: a multi-language program repair tool would have many more potential users than a single-language one, and could thus command far more investment. All existing program-repair benchmarks focus on a single language, and the use of distinct benchmark sets for different languages does not permit an apples-to-apples comparison of the tool's performance across languages.

Addressing this problem, we have created QuixBugs, the first multi-lingual parallel corpus of program repair benchmarks. The QuixBugs benchmark contains 40 programs translated into both Java and Python, each with a bug on a single line. Each buggy program is accompanied by passing and failing test cases, and a uniform driver for running any of the programs on their tests, in both Python and java.

2 Methodology

The programs in QuixBugs originated from the Quixey Challenge [3]. From 2011 to 2013, mobile app search startup Quixey[1] ran a challenge in which programmers were given an implementation of a classic algorithm with a bug on a single line, and had one minute to supply a fix, winning $100 upon successful completion. Their problems have proved

[1]Quixey announced its closure in February 2017, after raising over $150 million in funding.

an excellent dataset for program repair. Because they were developed as challenges for humans by people unaware of program repair, each buggy program is interesting, and the set as a whole has reduced potential for experimenter bias.

We manually translated all 40 buggy Python programs to Java. Several points of care were needed. Several challenges we faced arose from dealing with the differences between Java and Python's type systems. Python's use of heterogeneous collections caused trouble, as these were unnatural to translate into Java. Another point of care was required because different Python programs may use the same base data structure for different use cases (e.g.: a Python list may represent a list or a stack).

We created a test driver to serve as an interface to create the test input objects for Java. While standard JSON deserialization was sufficient for creating test objects in the form expected by most Python programs, the Java ones required inputs be statically converted to the parameter type required of the buggy Java method. The test driver uses Java's reflection to obtain these types and perform the conversion. Because the programs used inconsistent graph representations, we decided to hardcode the Python and Java testcases for problems with these inputs. We hope to find better methods of implementing these special testcases in future work. Some Python programs used fictitious libraries, particularly data structures such as min-heaps and ordered sets, as pseudocode; we supplied implementations of these. Finally, despite the relative verbosity of Java, we took care to maintain the property that all bugs were only on one line. Ultimately, we succeeded in translating all 40 buggy Python programs to Java programs with similar bugs.

Table 1 classifies the 40 programs by defect type. We tried to use the defect ontology of Tan et al [8]. However, we found it a poor match for the distribution of defects in QuixBugs. First, it does not have subcategories for especially common instances of its patterns (e.g.: off-by-one errors). Second, its ontology did not include nonlocal defect patterns such as swapped variables. We hence used our own classification.

We did not make it a goal to provide correct versions of each program. However, the original Quixey Challenge problems contained multiple corrected implementations of each Python program, which we have also included in QuixBugs.

3 Conclusion

This paper presents the QuixBugs benchmark that aims to facilitate future empirical study in automated program repair, particularly program repair tools that target multiple languages. Because this benchmark suite includes similar programs with the same bug across multiple languages, it allows for more objective measurement of how well a program repair tool is able to generalize across languages, as opposed to using distinct single-language benchmarks with different sets of bugs.

Table 1. Defect classes in the QuixBugs benchmark

Defect class	Count
Incorrect assignment operator	1
Incorrect variable	5
Incorect comparison operator	5
Missing condition	2
Missing/added +1	4
Variable swap	6
Incorrect array slice	2
Variable prepend	2
Incorrect data structure constant	2
Incorrect method called	1
Incorrect field dereference	1
Missing arithmetic expression	1
Missing function call	4
Missing line	4

The Java portion of QuixBugs was previously used by the Quixey Challenger, a prototype repair tool based on constraint-based search. We have begun developing a multi-language program repair tool based on our Cubix [2] framework for multi-language transformation. QuixBugs is available from https://github.com/jkoppel/QuixBugs.

Acknowledgments

The authors would like to thank Marshall Quander and Liron Shapira of Quixey for providing access to their data. This work is supported by NSF Grant No. 1122374.

References

[1] Dongsun Kim, Jaechang Nam, Jaewoo Song, and Sunghun Kim. 2013. Automatic Patch Generation Learned from Human-Written Patches. In *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 802–811.

[2] James Koppel. 2017. *"Incremental Parametric Syntax for Multi-Language Transformation"*. Master's thesis. MIT.

[3] Ryan Lawler. 2012. "How do you hire great engineers? Give them a challenge". https://gigaom.com/2012/01/19/quixey-challenge/. (2012). Accessed: 2017-07-16.

[4] Claire le Goues. 2014. https://bitbucket.org/clegoues/genprog4java. (2014).

[5] Claire Le Goues, Thanh Vu Nguyen, Stephanie Forrest, and Westley Weimer. 2012. Genprog: A Generic Method for Automatic Software Repair. *IEEE Transactions on Software Engineering* 38, 1 (2012), 54–72.

[6] Fan Long and Martin Rinard. 2016. Automatic Patch Generation by Learning Correct Code. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*.

[7] Matias Martinez and Martin Monperrus. 2016. ASTOR: A Program Repair Library for Java. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*. ACM, 441–444.

[8] Shin Hwei Tan, Jooyong Yi, Sergey Mechtaev, Abhik Roychoudhury, et al. 2017. "Codeflaws: A Programming Competition Benchmark for Evaluating Automated Program Repair Tools". In *Proceedings of the 39th International Conference on Software Engineering Companion*. IEEE Press, 180–182.