Mutation Testing of Quantum Programs Written in QISKit

Daniel Fortunato
daniel.b.fortunato@tecnico.ulisboa.pt
Faculty of Engineering of University of Porto & INESC-ID, Portugal

José Campos
jcmc@fe.up.pt
Faculty of Engineering of University of Porto, Portugal and LASIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal

Rui Abreu
rui@computer.org
Faculty of Engineering of University of Porto & INESC-ID, Portugal

ABSTRACT

There is an inherent lack of knowledge and technology to test a quantum program properly. In this paper, building on the definition of syntactically equivalent quantum operations, we investigated a novel set of mutation operators to generate mutants based on qubit measurements and quantum gates. To ease the adoption of quantum mutation testing, we further discuss QMutPy, an extension of the well-known and fully automated open-source mutation tool MutPy. To evaluate QMutPy's performance we conducted a case study on 11 real quantum programs written in the IBM's QISKit library. QMutPy has proven to be an effective quantum mutation tool, providing insight on the current state of quantum tests.

KEYWORDS

Quantum computing, Quantum software engineering, Quantum software testing, Quantum mutation testing

ACM Reference Format:
Daniel Fortunato, José Campos, and Rui Abreu. 2022. Mutation Testing of Quantum Programs Written in QISKit. In *44th International Conference on Software Engineering Companion (ICSE '22 Companion), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3510454.3528649

1 INTRODUCTION

While the fast approaching universal access to quantum computers is bound to break several computation limitations that have lasted for decades, it is also bound to pose significant challenges in, e.g., *software testing*. *Testing* refers to the execution of the software *in vitro* environments that replicate real scenarios to ascertain their correct behavior [3]. Despite that, in the classical computing realm, *testing* has been extensively investigated, and several approaches and tools have been proposed [1], Such approaches for Quantum Programs (QPs) are still in their infancy [10]. It is worth noting that (i) QPs are much harder to develop than classic programs and therefore programmers, mostly familiar with the classic world, are more likely to make mistakes in the counter-intuitive quantum programming one [6], and (ii) QPs are necessarily probabilistic and impossible to examine without disrupting execution or without compromising their results [7]. Thus, ensuring the correct implementation of QPs is even more challenging in the quantum computing realm [4].

ICSE '22 Companion, May 21–29, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9223-5/22/05.
https://doi.org/10.1145/3510454.3528649

Mutation testing [8] has been shown to be an effective technique in improving testing practices, hence helping in guaranteeing program correctness. Big tech companies, such as Facebook and Google, have conducted several studies [9] advocating for mutation testing and its benefits. The general principle underlying mutation testing is that the *bugs* considered to create versions of the program represent realistic mistakes that programmers often make. Such *bugs* are deliberately seeded into the original program by simply applying syntactic changes to the program to create a set of *buggy* programs called mutants. To assess the effectiveness of a test suite, these mutants are executed against the program's test suite. If the result of running a mutant is different from the result of running the original program, the mutant is considered *detected* or *killed*.

In this paper, we investigate the application of mutation testing on real QPs. We focus our investigation on the most popular open-source full-stack library for quantum computing [2], IBM's Quantum Information Software Kit (QISKit)[1]. More specifically, in this paper, we discuss: (1) A set of five novel mutation operators, leveraging the notion of syntactically equivalent gates, tailored for QPs; (2) A novel Python-based toolset named QMutPy[2] that automatically performs mutation testing for QPs written in the QISKit's full-stack library. (3) An empirical evaluation of QMutPy's effectiveness and efficiency on 11 real QPs. Our results suggest that QMutPy can generate fault-revealing quantum mutants, and it surfaced several issues in the test suites of real QPs.

2 MUTATION TESTING OF QPS

2.1 Quantum Mutation Operators

Similar to classic programs, a QP is fundamentally a circuit in which quantum bits (*qubits*) are initialized and go through a series of operations that change their state. These operations are commonly known and referenced as *quantum gates*.

At the time of writing this paper, QISKit v0.29.0 provides support to more than 50 quantum gates. This includes single-qubit gates (e.g., `h` gate), multiple-qubit gates (e.g., `cx` gate) and composed gates, or circuits (e.g., `QFT` circuit). Given their importance on the execution and result of a QP, as a simple typo on the name of the gate could cause *bugs* that developers may not be aware of, our set of mutation operators to generate faulty versions of QPs is based on single- and multi-qubit *quantum gates*, in particular, *syntactically equivalent gates*. We argue that our quantum mutants match real world *bugs* as (1) Liu et al. [5] described quantum mutation to be useful to assess the correct behavior of QPs, and (2) 3 out of the 8 common *bug* patterns in QISKit programs described by Zhao et al. [11] are related to quantum gates, as is the majority of our mutation operators.

[1] QISKit homepage, https://qiskit.org, accessed March/2022.
[2] QMutPy source code, https://github.com/danielfobooss/mutpy, accessed March/2022.

Formally, a gate g is considered *syntactically equivalent* to a gate j if and only if the number and the type of arguments[3] required by both g and j are the same. At the time when we performed our experiment, we had identified 40 gates that had syntactical equivalents. Note that these gates do not perform or compute the same operation; they are simply used in the same manner and require the same number and type of arguments. We briefly describe the five quantum mutation operators investigated in this paper:
— *Quantum Gate Replacement (QGR)*: for each quantum gate function call (e.g., `circuit.x()`), replaces it with all syntactically equivalent gates, e.g., `circuit.h()`, one at a time.
— *Quantum Gate Deletion (QGD)*: deletes a call to a quantum gate.
— *Quantum Gate Insertion (QGI)*: for each quantum gate in the source code, inserts a call to all syntactically equivalent gates one at a time.
— *Quantum Measurement Insertion (QMI)*: adds a call to the `measure` function for each quantum gate call.
— *Quantum Measurement Deletion (QMD)*: removes each measurement from a QP, one at a time.

2.2 QMutPy

We built QMutPy on top of MutPy[4], an already well adopted mutation tool. Given a Python program P, its test suite T, and a set of mutation operators M, QMutPy automatically performs the following workflow: (1) Loads P's source code and its test suite; (2) Executes T on the original source code; (3) Applies M and generates all mutant versions of P; (4) Executes T on each mutant and provides a summary of the results either as a `yaml` or `html` report.

3 PRELIMINAR EMPIRICAL STUDY

Our goal is to analyze the quality and resilience of test suites designed to verify QPs. As mentioned before, the idiosyncrasies underlying QPs (e.g., superposition, entanglement) makes testing far from trivial. We argue that QMutPy's mutants can be used as benchmarks to assess the quality of tests designed to verify QPs. We considered 11 QPs written in the IBM's QISKit library that range from 80 to 443 lines of code (245 on average).

Our set of quantum mutation operators generated a total of 696 mutants for the 11 QPs, of which 325 (46.7%) were killed by the programs' test suites. QGI, the mutation operator that generated more mutants (328), had a ratio of 102 killed mutants, followed by QGR 170 killed mutants out of 300 generated, QMI 27 out of 28, QGD 18 out of 28, and QMD 8 out of 12. QGD, QGR, QGI, and QMD mutants are killed more often by *test assertions* than by *crashes*. We also observed that QMI mutants, as expected, are killed by *crashes* only. The reason is that QISKit does not have a fail-safe mechanism for inserting measurements. When a measurement operation is inserted in a random position, the circuit may become unprocessable and an *unexpected exception* is thrown. However, developing better approaches to reduce the number of design errors of QMI mutants remains as future work. The non-killed mutants either survived to the test suites (307, 44.1%), were not even exercised by the test suites (2 QMD mutants, 0.3%), or resulted in a timeout (62, 8.9%).

Regarding results at program level, on average, our set of mutation operators mutated 4 lines of code (1.4% of all lines) and generated 14 mutants per mutated line. The mutation score achieved by all programs' test suites was, on average, 57.7%. In detail, vqc's and vqe's test suite failed to kill any of the generated mutants. The single mutant generated for vqe (QMD) timeout and one of the two generated mutants generated for vqc is not exercised by the program's test suite. Recall that non-exercised mutants would never be killed by any test as the mutated code is never executed. qsvm's test suite killed the single generated mutant (QMD), hhl's 1 out of 2 (the survived one is not exercised by the test suite), simon's 32 out of 47, grover_optimizer's 2 out of 52 (50 timeout), bernstein_vazirani's 74 out of 93, deutsch_jozsa's 66 out of 93, grover's 17 out of 93, iqpe's 82 out of 105 (4 timeout), and shor's 50 out of 207 (7 timeout).

To verify whether quantum mutants are not killed by chance but due to tests tailored to verify specific quantum behaviors, we conducted a small experiment on two QPs, i.e., shor and grover. We first removed all test assertions from shor and grover's test suite, then re-ran our mutation analysis on each QP, and finally re-computed mutation scores. The mutation scores achieved in this experiment by each programs' test suite dropped 42.6%. This shows the intention of testing a specific quantum behavior is the main reason tests kill most quantum mutants.

4 CONCLUSIONS

In this paper, we discussed a mutation-based technique to test QPs, coined QMutPy, which is capable of mutating QPs written QISKit. This is a first attempt to perform mutation testing on QPs with a tool that is easy to use and works at scale. Furthermore, QMutPy offers classic and more quantum mutation operators than the approaches/tools proposed in the literature. To demonstrate the effectiveness of QMutPy, we have carried out an empirical study with 11 real QPs. We observed several issues that may lead to future failures, e.g., non-optimal code coverage and low mutation scores. As for future work, we plan to extend QMutPy with other mutation operators, offer it to other quantum frameworks (e.g., Cirq and Q#), and run our mutation analysis on real quantum computers.

ACKNOWLEDGMENTS This work was supported in part by FCT/MCTES through projects ref. PTDC/CCI-COM/29300/2017 and CMU/TIC/0064/2019, and the research units LASIGE (ref. UIDB/00408/2020 and UIDP/00408/2020) and INESC-ID (ref. UIDB/50021/2020).

[3]Optional arguments are not taken into consideration.
[4]MutPy source code, https://github.com/mutpy/mutpy, accessed March/2022.

REFERENCES

[1] M. Moein Almasi, Hadi Hemmati, Gordon Fraser, Andrea Arcuri, and Jundefinednis Benefelds. 2017. An Industrial Evaluation of Unit Test Generation: Finding Real Faults in a Financial Application. In *Proceedings of the 39th ICSE-SEIP*.
[2] Mark Fingerhuth, Tomáš Babej, and Peter Wittek. 2018. Open source software in quantum computing. *PLOS ONE* (2018).
[3] Gordon Fraser and José Miguel Rojas. 2019. *Software Testing*. Springer International Publishing, Cham, 123–192. https://doi.org/10.1007/978-3-030-00262-6_4
[4] Yipeng Huang and Margaret Martonosi. 2018. QDB: from quantum algorithms towards correct quantum programs. *arXiv preprint arXiv:1811.05447* (2018).
[5] P. Liu, S. Hu, M. Pistoia, C. R. Chen, and J. M. Gambetta. 2019. Stochastic Optimization of Quantum Programs. *Computer* 52, 6 (2019), 58–67.
[6] Andriy V. Miranskyy and Lei Zhang. 2018. On Testing Quantum Programs. *CoRR* abs/1812.09261 (2018). arXiv:1812.09261 http://arxiv.org/abs/1812.09261
[7] Michael A. Nielsen and Isaac L. Chuang. 2010. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press.
[8] Goran Petrović, Marko Ivanković, Gordon Fraser, and René Just. 2021. Does Mutation Testing Improve Testing Practices?. In *Proc. of the 43rd IEEE/ACM ICSE*.
[9] Goran Petrović, Marko Ivanković, Gordon Fraser, and René Just. 2021. Practical Mutation Testing at Scale: A view from Google. *IEEE TSE* (2021).
[10] Jianjun Zhao. 2020. Quantum Software Engineering: Landscapes and Horizons. arXiv:2007.07047 [cs.SE]
[11] Pengzhan Zhao, Jianjun Zhao, and Lei Ma. 2021. Identifying Bug Patterns in Quantum Programs. In *Proc. of the 2nd Q-SE*.