MoFuzz: A Fuzzer Suite for Testing Model-Driven Software Engineering Tools – Summary

Hoang Lam Nguyen[1] Nebras Nassar[2] Timo Kehrer[1] Lars Grunske[1]

Abstract: Fuzzing or fuzz testing is an established technique that aims to discover unexpected program behavior (e. g., bugs, vulnerabilities, or crashes) by feeding automatically generated data into a program under test. However, the application of fuzzing to test Model-Driven Software Engineering (MDSE) tools is still limited because of the difficulty of existing fuzzers to provide structured, well-typed inputs, namely models that conform to typing and consistency constraints induced by a given meta-model and underlying modeling framework. We present three different approaches for fuzzing MDSE tools: A graph grammar-based fuzzer and two variants of a coverage-guided mutation-based fuzzer working with different sets of model mutation operators. Our evaluation on a set of real-world MDSE tools shows that our approaches can outperform both standard fuzzers and model generators w.r.t. their fuzzing capabilities. Moreover, we found that each of our approaches comes with its own strengths and weaknesses in terms of code coverage and fault finding capabilities, thus complementing each other and forming a fuzzer suite for testing MDSE tools.

Keywords: Model-Driven Software Engineering; Modeling Tools; Fuzzing; Automated Model Generation; Eclipse Modeling Framework

1 Summary

Fuzzing (also known as *fuzz testing*) automatically generates a large number of inputs and feeds them to the program under test to discover unexpected program behavior and evaluate the program's reliability. In our work, we investigate the fuzzing of Model-Driven Software Engineering (MDSE) tools which are based on the Eclipse Modeling Framework (EMF). Fuzzing MDSE tools is a challenging task, since (i) the test inputs must adhere to complex input constraints (e. g., well-typedness and valid multiplicities w.r.t. the input meta model) in order to pass the initial syntactic and semantic validation stages of the input processing pipeline, and (ii) the generated input models must be interesting/diverse enough to exercise a variety of code paths.

Building upon recent advances in automated model generation and structure-aware fuzzing, we propose three different fuzzers as part of our fuzzer suite MoFuzz [Ng20]: a graph-grammar based fuzzer and two mutation-based approaches. The graph grammar-based fuzzer is based on the recently introduced EMF Model Generator [Na20], which is able to efficiently generate large, properly-typed EMF models with valid multiplicities. Conceptually,

[1] Humboldt-Universität zu Berlin, Germany, {nguyehoa,kehrer,grunske}@informatik.hu-berlin.de
[2] Philipps-Universität Marburg, Germany, nassarn@informatik.uni-marburg.de

doi:10.18420/SE2021_29

the fuzzer first translates the meta-model into a constructive language specification (i. e., the grammar), which is then leveraged to generate models in a two-phased approach. First, the *model increase* phase creates model elements without violating upper multiplicity bounds. Then, the *model completion* phase completes the intermediate model to a valid EMF model. Overall, the graph-grammar based fuzzer attempts to *broadly* explore the space of valid instance models in an efficient manner. The mutation-based fuzzers are based on a widely used technique in automated fault detection, namely coverage-guided fuzzing (CGF) [LZZ18], which we adapt to the domain of MDSE as follows. First, a random set of seed models is generated using automated model generation techniques to initialize the input queue. Afterwards, both approaches continuously select an input model from the queue, apply model-based mutations on it, and retain the mutated input only if it increases coverage. The goal is to incrementally evolve the inputs in the queue to exercise deep paths. While both fuzzers essentially employ mutations that add, delete, or change model elements, one uses generic mutation operators based on the EMF Edit API, whereas the other automatically derives consistency-preserving mutation operators from the meta-model [Ke16].

Our implementation of MoFuzz builds upon JQF [PLS19], a feedback-directed fuzz testing framework for Java. We have evaluated MoFuzz gainst the Zest algorithm implemented by JQF, and the EMF random instantiator [At15] on a set of real-world MDSE tools. The results of our evaluation indicate that MoFuzz can improve code coverage as well as the number of exposed crashes when fuzzing MDSE tools. In terms of coverage, the graph grammar-based fuzzer of MoFuzz performed the best, while the mutation-based fuzzer using EMF Edit API mutations triggered the most crashes.

Bibliography

[At15] AtlanMod: EMF Random Instantiator. https://github.com/atlanmod/mondo-atlzoo-benchmark/tree/master/fr.inria.atlanmod.instantiator/, 2015. Accessed: November 24, 2020.

[Ke16] Kehrer, Timo; Taentzer, Gabriele; Rindt, Michaela; Kelter, Udo: Automatically Deriving the Specification of Model Editing Operations from Meta-Models. In: 9th International Conference on Theory and Practice of Model Transformations (ICMT). pp. 173–188, 2016.

[LZZ18] Li, Jun; Zhao, Bodong; Zhang, Chao: Fuzzing: a survey. Cybersecurity, 1(1):6, 2018.

[Na20] Nassar, Nebras; Kosiol, Jens; Kehrer, Timo; Taentzer, Gabriele: Generating Large EMF Models Efficiently - A Rule-Based, Configurable Approach. In: 23rd International Conference on Fundamental Approaches to Software Engineering (FASE). Springer, pp. 224–244, 2020.

[Ng20] Nguyen, Hoang Lam; Nassar, Nebras; Kehrer, Timo; Grunske, Lars: MoFuzz: A Fuzzer Suite for Testing Model-Driven Software Engineering Tools. In: 35th IEEE/ACM International Conference on Automated Software Engineering (ASE). 2020.

[PLS19] Padhye, Rohan; Lemieux, Caroline; Sen, Koushik: JQF: Coverage-Guided Property-Based Testing in Java. In: 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA). pp. 398–401, 2019.