Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
Electronic Notes in Theoretical Computer Science 253 (2009) 57–75
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Using Heuristics to Automate Parameter Generation for Benchmarking of Java Methods

Michael Kuperberg[1] Fouad Omri[2]

Chair for Software Design and Quality
Institute for Program Structures and Data Organisation
Faculty of Informatics, Universität Karlsruhe (TH)

Abstract

Automated generation of method parameters is needed in benchmarking scenarios where manual or random generation of parameters are not suitable, do not scale or are too costly. However, for a method to execute correctly, the generated input parameters must not violate implicit semantical constraints, such as ranges of numeric parameters or the maximum length of a collection. For most methods, such constraints have no formal documentation, and human-readable documentation of them is usually incomplete and ambiguous. Random search of appropriate parameter values is possible but extremely ineffective and does not pay respect to such implicit constraints. Also, the role of polymorphism and of the method invocation targets is often not taken into account. Most existing approaches that claim automation focus on a single method and ignore the structure of the surrounding APIs where those exist. In this paper, we present HeuriGenJ, a novel heuristics-based approach for automatically finding legal and appropriate method input parameters and invocation targets, by approximating the implicit constraints imposed on them. Our approach is designed to support systematic benchmarking of API methods written in the Java language. We evaluate the presented approach by applying it to two frequently-used packages of the Java platform API, and demonstrating its coverage and effectiveness.

Keywords: Heuristics, parameter generation, exception handling, automated benchmarking, constraint approximation

1 Introduction

Most software applications developed today build on object-oriented languages and execution platforms. For example, the Java Virtual Machine executes Java bytecode, to which the Java programming language and other programming languages are compiled. For Java, the building blocks of such applications are classes, which contain methods and fields. The functional properties (e.g. correctness) and extra-functional properties (e.g. performance) of methods are subject of ongoing research,

[1] Email: mkuper@ipd.uka.de
[2] Email: omri@ipd.uka.de

doi:10.1016/j.entcs.2009.09.028

and analysis of these properties must consider the impact of method input parameters. Also, the state of the objects and class instances whose methods are invoked must be considered.

For example, in benchmarking, the input parameters often have a strong impact on the method performance, so different parameters must be studied. This task quickly becomes too expensive for manual implementation if the number of methods is very high, as it is often the case in the application programming interfaces (APIs): the Java platform API contains several thousands of methods. However, there exists no automated API benchmarking tool or strategy for Java APIs. In this paper, Java API denotes *any* API compiled to and accessible from Java bytecode; we explicitly refer to the *Java* ***platform*** *API* when the functionality provided by the Java Runtime Environment is meant.

Where manual generation of method parameters is not suitable, randomised approaches are often tried, but they become ineffective where the potential parameter space is too large. Also, existing randomised approaches mostly focus on testing-oriented cases, i.e. on finding cases where software's behaviour *deviates* from the expected, specified targets. In contrast to maximising failure occurrence, benchmarking needs to find parameters that do *not* deviate from expected execution w.r.t. exceptions and errors. Also, software testing does not need to recover or to learn from failed parameters, while in benchmarking, failures must be minimised as much as possible to achieve good coverage.

If a method requires input parameters, they must be provided in accordance with their static types in the method's signature, e.g. for interface-typed parameters, an instance of a class implementing that interface must be passed. In addition, implicit semantical requirements for these parameters exist: for example, the method `java.lang.String. substring(int beginIndex)` throws an `IndexOutOfBoundsException` for an instance `str` if `beginIndex < 0` or if `beginIndex` $\geq$ `str.length()`.

In the cases where such requirements are given, if at all, they are described informally by humans and for humans, and thus cannot be evaluated by tools due to the complexity and general ambiguity of human language. Also, there are no formal specifications that can be used by an automated approach. Guessing an appropriate value using a random search is intractable given the large range of possible values that could be generated for each single parameter; for the above example, the parameter `beginIndex` has a range of 2^{32} different values. The few existing approaches that claim automation of parameter generation focus on a single method and ignore the structure of the surrounding APIs where those exist.

The contribution of this paper is a novel self-correcting approach for the automatic generation of input parameters for Java methods, based on formally-defined heuristics. The heuristics help to find parameters which can be used in meaningful benchmarks. The presented approach detects inappropriate methods arguments on the basis of thrown exceptions, automatically approximates underlying exception causes using novel heuristics and recovers them by generating new and appropriate input parameters. The generation of the parameter values for a method is not based