Desenvolvimento de um Gerador de Programas Aleatórios em Java

Luiz Felipe Kraus
Instituto Federal de Santa Catarina
Caçador, Santa Catarina, Brasil
luiz.k1999@aluno.ifsc.edu.br

Bruno Schafaschek
Instituto Federal de Santa Catarina
Caçador, Santa Catarina, Brasil
bruno.sc11@aluno.ifsc.edu.br

Samuel da Silva Feitosa
Instituto Federal de Santa Catarina
Caçador, Santa Catarina, Brasil
samuel.feitosa@ifsc.edu.br

ABSTRACT

With great advances in the computer science area where technological systems are becoming more and more complex, tests are hard to perform. The problem is even more serious in critical systems, such as flight control or nuclear systems, where an error can cause catastrophic damage in our society. Currently, two techniques are often used for software validation: testing and software verification. This project aims the testing area, generating random programs to be used as input to property-based tests, in order to detect errors in systems and libraries, minimizing the possibility of errors. More specifically, Java programs will be automatically generated from existent classes and interfaces, considering all syntactic and semantic constraints of the language.

KEYWORDS

Geração de Programas Aleatórios, Teste Baseado em Propriedades, Java

1 INTRODUÇÃO

Atualmente, Java é uma das linguagens de programação mais utilizadas no mundo [1]. Por se tratar de uma linguagem de propósito geral, concorrente, fortemente tipada e orientada a objetos, tem sido utilizada em muitos projetos de média e grande escala. Nestes casos, aplicações tendem a alcançar níveis de complexidade, onde apenas revisão de código e testes manuais não são suficientes para garantir qualidade no desenvolvimento de software.

A área de pesquisa em testes de sistemas tem avançado cada vez mais, fornecendo metodologias e ferramentas que permitem aos desenvolvedores obter maiores garantias de que o sistema realiza as atividades em que se propõe, e evitar a liberação de módulos dos programas com erros de execução. Para testar atividades rotineiras internas em códigos orientados a objetos de forma automatizada, o mais comum é se utilizar de *testes unitários*, onde o programador define uma série de casos de testes que cada elemento do sistema deve respeitar, e todos esses casos de teste são verificados antes do sistema ser enviado para o ambiente de produção [2].

Entretanto, o uso de testes unitários depende exclusivamente da capacidade dos desenvolvedores na modelagem desses casos de teste. Outra abordagem que vem ganhando espaço para a realização de testes automatizados, é conhecida como *testes baseados em propriedades*, onde o programador define regras gerais (propriedades) que o sistema deve respeitar, e o mecanismo de testes faz a geração aleatória de dados de entrada e verifica se a propriedade é respeitada [3]. Desta forma, o sistema é testado com um conjunto infinito de variações, permitindo que mais erros sejam encontrados ainda em tempo de desenvolvimento.

Neste sentido, este projeto tem o intuito de dar um passo a frente na geração de dados para testes de propriedades, gerando aleatoriamente programas Java completos e bem tipados, a partir de classes e interfaces definidas em pacotes ou bibliotecas de código, e em regras formais do sistema de tipos da linguagem. Estes programas completos podem ser utilizados como entrada para propriedades mais gerais, responsáveis por garantir o funcionamento do sistema, ou ainda para testes do compilador em si, que também pode conter falhas, que podem ser encontradas a partir da execução de um número grande de diferentes programas.

Para possibilitar o desenvolvimento deste projeto, será necessário cumprir diversas etapas:

- Especificação das regras de tipo para os principais construtores sintáticos da linguagem Java.
- Obtenção de informações a respeito das classes/interfaces alvos do mecanismo de geração.
- Geração de código Java com os construtores sintáticos selecionados e respeitando as regras de tipo.
- Definição de propriedades e execução de testes para validar o mecanismo desenvolvido.

Para que seja possível desenvolver essas etapas, já foi definido o aparato ferramental necessário, o qual inclui o uso da biblioteca *Reflection*[1] para obtenção de informações sobre código Java, a biblioteca *JavaParser*[2] que permite manipular construtores sintáticos e exportar código real, e a biblioteca *JQWik*[3] que provê geradores para construções básicas e gerencia a execução dos testes das propriedades definidas.

O restante deste texto está organizado da seguinte forma: a seção 2 apresenta os conceitos de geração de código baseadas em tipos e de testes baseados em propriedades. Na seção 3 é apresentada a arquitetura utilizada para desenvolver este projeto. A seção 4 apresenta os resultados parciais já obtidos. As seções 5 e 6 encerram este documento apresentando os trabalhos relacionados e considerações finais.

2 REFERENCIAL TEÓRICO

2.1 Geração de Código Baseada em Tipos

A criação de ferramentas para gerar programas aleatórios válidos é uma tarefa árdua, uma vez que é difícil implementar um gerador para programas que são aceitos pelo compilador. A produção de um caso de teste válido e útil deve respeitar uma série de restrições, como por exemplo a correta sintaxe e as regras do sistema de tipos em linguagens estaticamente tipadas [4].

[1] https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/package-summary.html
[2] www.javaparser.org
[3] www.jqwik.com

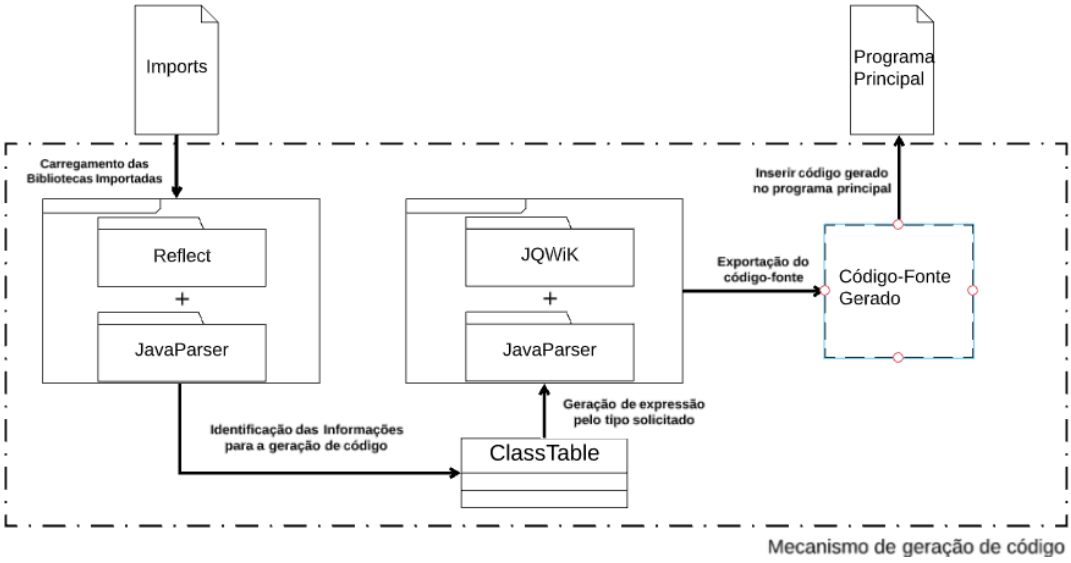

Figure 1: Arquitetura do sistema de geração.

Uma abordagem interessante é utilizar as regras formais do sistema de tipos para gerar apenas programas bem tipados e armazená-los em árvores de sintaxe abstrata (ASTs) da linguagem alvo [5]. Tendo as ASTs em mãos, basta exportá-las para código fonte da linguagem escolhida.

2.2 Teste Baseado em Propriedades

Teste baseado em propriedades é uma abordagem para realização de testes em sistemas onde o programador define uma série de condições (propriedades) que o sistema deve respeitar, e essas condições são verificadas a partir de dados de entrada gerados aleatoriamente. A ferramenta precursora para esta tarefa é o QuickCheck, que permite efetuar testes baseados em propriedades na linguagem Haskell [3].

O uso desta abordagem auxilia o desenvolvedor a encontrar *bugs* que poderiam passar despercebidos apenas com inspeção do código, uma vez que o mecanismo de geração aleatório de entradas permite a execução do trecho do programa com diversas entradas, aumentando a probabilidade de encontrar um erro em tempo de produção.

3 ARQUITETURA

Para o desenvolvimento deste projeto, a partir das tecnologias escolhidas, foi definida a arquitetura apresentada na Figura 1.

O fluxo de execução do mecanismo de geração de código apresentado na arquitetura proposta inicia com a definição de uma lista de bibliotecas (imports) no esqueleto de uma classe principal. Tendo a lista de bibliotecas definidas, a primeira etapa efetua o carregamento das informações de cada classe (construtores, atributos, métodos, etc.) fazendo o uso das ferramentas *Java Reflect* e *JavaParser*, resultando em uma classe chamada de *ClassTable*. Na segunda etapa, utilizam-se as informações disponíveis na *ClassTable* para seleção de um tipo válido (tipo primitivo ou definido pelo usuário nas bibliotecas importadas), e então é realizada a geração aleatória de combinações sintáticas válidas da linguagem, utilizando as bibliotecas *JQWik* e *JavaParser*, respeitando as restrições impostas pelo sistema de tipos, ao final gerando um meta-programa em uma AST. Na última etapa, a AST gerada é compilada para código Java real, o qual é embutido no esqueleto da classe principal citado anteriormente. O mecanismo de geração atual é composto de diversas classes, cada qual implementando determinadas instruções da linguagem Java:

- JRGBase: implementa a geração de tipos primitivos como int, float, char, etc.
- JRGCore: faz a geração de expressões básicas, como por exemplo a criação de objetos, acesso a atributos, invocação de métodos, etc.
- JRGStmt: faz a geração de *statements*, incluindo blocos de código, estruturas condicionais e de repetição.

Os códigos-fonte gerados pelo mecanismo proposto são utilizados como entrada para sistemas de testes, que podem ser tanto testes unitários, testes baseados em propriedades, ou testes automatizados para testar a infraestrutura (compilador, máquina virtual, etc.) dos kits de desenvolvimento Java.

4 RESULTADOS PARCIAIS

O projeto iniciou a partir de uma pesquisa a respeito do aparato ferramental para desenvolvimento do projeto, considerando as necessidades de examinar ou inspecionar código Java, representar árvores de sintaxe abstrata (ASTs) e exportá-las para código, e *frameworks* para realização de testes baseados em propriedades. Com este estudo, foram selecionadas as ferramentas *Reflection*, *JavaParser* e *JQWik*.

Na sequência, foram definidas algumas regras de tipos para construções básicas do Java, usando como base uma formalização de

um subset de Java [6]. Por exemplo, a regra abaixo formaliza o tipo de uma expressão que cria um novo objeto.

$$\frac{\mathit{constr}(\mathrm{C}) = \bar{\mathrm{D}}\ \bar{\mathrm{f}} \quad \Gamma \vdash \bar{\mathrm{e}} : \bar{\mathrm{C}} \quad \bar{\mathrm{C}} <: \bar{\mathrm{D}}}{\Gamma \vdash \text{new } \mathrm{C}(\bar{\mathrm{e}}) : \mathrm{C}} \text{ [T-New]}$$

O processo de geração considera a regra de tipo de traz para a frente, iniciando com a definição aleatória do tipo C, para o qual será criado um novo objeto, para posteriormente gerar os parâmetros para o construtor $\bar{\mathrm{e}}$, respeitando a lista de construtores definidas na classe. A função *constr*(C) traz um construtor válido para a classe C, e a operação $\bar{\mathrm{C}} <: \bar{\mathrm{D}}$ verifica a relação de subtipos entre as classes.

Este projeto está em fase de desenvolvimento, onde expressões estão sendo geradas individualmente para cada regra de tipo. Na sequência deste projeto, estas gerações individuais serão combinadas para a geração de um programa Java completo.

5 TRABALHOS RELACIONADOS

Existem diversos artigos que definem técnicas para a geração de códigos aleatórios para o contexto de testes baseados em propriedades. Um dos trabalhos pioneiros utiliza a biblioteca QuickCheck e gera código Haskell [5]. Também existem soluções propostas especificamente para a geração aleatória de código Java [4, 7, 8].

6 CONSIDERAÇÕES FINAIS

No atual estágio deste projeto, já foram definidas as ferramentas que viabilizam a extração e exportação de código Java, bem como a aplicação de testes baseados em propriedades. Além disso, já foram executados diversos testes com todas as ferramentas, como uma medida de verificação de adequação das mesmas para o projeto proposto. Algumas etapas do processo de formalização e geração de código também já foram desenvolvidas, como por exemplo a formalização de regras de tipos para expressões básicas da linguagem Java, e a geração de expressões individuais para estas regras.

Considerando os estudos já realizados, acredita-se que este projeto apresenta viabilidade técnica e contribuições importantes tanto para a área prática de testes de software, como para a área de formalização de sistemas e linguagens de programação. A sequência deste projeto compreende a continuidade da definição dos mecanismos de geração, as aplicações destes programas gerados para o teste de programas e bibliotecas Java, e a definição formal do mecanismo de geração e do sistema de tipos dos construtores sintáticos utilizados.

AGRADECIMENTOS

Este projeto é financiado parcialmente pelo CNPq.

REFERÊNCIAS

[1] Stephen Cass. As principais ling. de programação de 2019. IEEE Spec., Set 2019. URL https://spectrum.ieee.org/computing/software/the-top-programming-languages-2019.

[2] Jeff Langr, Andy Hunt, and Dave Thomas. *Pragmatic Unit Testing in Java 8 with JUnit.* Pragmatic Bookshelf, 1st edition, 2015. ISBN 1941222595.

[3] Koen Claessen and John Hughes. Quickcheck: A lightweight tool for random testing of haskell programs. *SIGPLAN Not.*, 35(9):268–279, September 2000. ISSN 0362-1340. doi: 10.1145/357766.351266. URL https://doi.org/10.1145/357766.351266.

[4] Samuel Feitosa, Rodrigo Ribeiro, and Andre Du Bois. A type-directed algorithm to generate random well-typed java 8 programs. *Science of Computer Programming*, 196:102494, 2020. ISSN 0167-6423. doi: https://doi.org/10.1016/j.scico.2020.102494. URL http://www.sciencedirect.com/science/article/pii/S0167642320301039.

[5] Michał H. Palka. Testing an optimising compiler by generating random lambda terms, Jun 2003. URL https://core.ac.uk/download/pdf/70594393.pdf.

[6] G. M. Bierman, M. J. Parkinson, and A. M. Pitts. An imperative core calculus for java and java with effects. Technical report, 2003.

[7] Tristan O. R. Allwood and Susan Eisenbach. Tickling Java with a feather. *Electron. Notes Theor. Comput. Sci.*, 238(5):3–16, October 2009. ISSN 1571-0661. doi: 10.1016/j.entcs.2009.09.037. URL http://dx.doi.org/10.1016/j.entcs.2009.09.037.

[8] Casey Klein, Matthew Flatt, and Robert Bruce Findler. Random testing for higher-order, stateful programs. In *Proc. of the ACM Int. Conf. on Object Oriented Prog. Systems Languages and Applications*, OOPSLA '10, pages 555–566, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0203-6. doi: 10.1145/1869459.1869505. URL http://doi.acm.org/10.1145/1869459.1869505.