Desenvolvimento de uma Ferramenta para Teste Diferencial de Compiladores Usando Códigos Gerados Aleatoriamente*

Bruno Schafaschek Coelho[1], **Luiz Felipe Krauz**[1], **Samuel da Silva Feitosa**[1]

[1] Instituto Federal de Santa Catarina (IFSC) – Caçador – SC – Brasil

{bruno.sc11, luiz.k1999}@aluno.ifsc.edu.br, samuel.feitosa@ifsc.edu.br

Abstract. *It is notable that software projects are becoming more complex and extensive, which depend mostly on the compiler or interpreter of the programming language chosen for their development. In this sense, how is it possible to guarantee the quality and reliability of these development tools? One of the possibilities is to run tests exhaustively, identifying and correcting errors, until we have some certainty that the code produced is free of bugs. In this way, this project proposes the development of a tool that applies generated randomly test cases and compares the results with different Java compilers.*

Resumo. *É notável que os projetos de software estão se tornando mais complexos e extensos, os quais dependem em grande parte do compilador ou interpretador da linguagem de programação escolhida para o seu desenvolvimento. Neste sentido, como é possível garantir a qualidade e confiabilidade dessas ferramentas de desenvolvimento? Uma das possibilidades é executar testes exaustivamente, identificando e corrigindo erros, até que se tenha certa segurança de que o código produzido esteja livre de* bugs*. Sendo assim, este projeto propõe o desenvolvimento de uma ferramenta que aplique casos de teste gerados aleatoriamente e compare os resultados com os diferentes compiladores de Java.*

1. Introdução

Atualmente Java está entre as linguagens de programação mais utilizadas no mundo [Cass 2019], sendo uma linguagem orientada a objetos e fortemente tipada, que vem sendo aplicada em projetos de diversos portes. Por ser utilizada em projetos grandes e em sistemas críticos, torna-se importante garantir que não ocorram erros de execução, uma vez que qualquer problema poderia causar danos irreversíveis.

Sendo assim, para minimizar possíveis erros em projetos de software é de grande importância que diversos testes sejam executados. Isso não é diferente na área de desenvolvimento de compiladores. Soma-se a isso a existência de diferentes implementações (compiladores e máquinas virtuais) para a linguagem Java. Entretanto, somente através da utilização de testes conduzidos por pessoas não é possível garantir eficácia e qualidade, já que os casos de testes podem ficar incompletos devido a própria limitação humana [Palka et al. 2011]. Além disso, é praticamente impossível que uma equipe de testes consiga testar efetivamente dezenas de milhares de linhas de códigos.

Neste contexto, este projeto busca responder ao seguinte questionamento: *É possível detectar bugs em compiladores Java com maior cobertura de código, através*

*Este trabalho encontra-se em fase de desenvolvimento.

de teste diferencial, usando programas gerados aleatoriamente? Estão previstas duas etapas para obter esta resposta. Primeiro, é proposto a adequação do gerador de códigos aleatórios [Kraus et al. 2021] desenvolvido previamente pelos autores deste artigo, o qual é capaz de construir programas automaticamente a partir de classes e interfaces pré-existentes, para atuar no contexto de teste diferencial de compilares [McKeeman 1998]. Segundo, será desenvolvida uma ferramenta capaz de testar propriedades em diferentes implementações da linguagem Java, procurando possíveis diferenças entre elas, as quais indicariam uma falha no processo de desenvolvimento. Para este trabalho, serão consideradas as propriedades de compilação e comportamento.

O restante do texto está estruturado da seguinte forma: Na Seção 2 encontram-se os trabalhos relacionados, a Seção 3 apresenta a técnica de teste diferencial de compiladores, a Seção 4 é onde se define a geração de código aleatório, e, por fim, as Seções 5 e 6 apresentam os resultados parciais e os resultados esperados, respectivamente.

2. Trabalhos Relacionados

A biblioteca QuickCheck [Claessen and Hughes 2000] é uma das precursoras na área de testes baseados em propriedades, a qual utiliza dados gerados aleatoriamente como entrada. Utilizando esta biblioteca, Palka et al. [Palka et al. 2011] desenvolveu um gerador de código Haskell para efetuar o teste diferencial do compilador desta linguagem. Este trabalho serve de inspiração para a proposta deste artigo, o qual considera a linguagem Java, ao invés do Cálculo Lambda, envolvendo uma complexidade maior em termos de construções sintáticas e semânticas.

Outros trabalhos na área de teste diferencial de compiladores também foram desenvolvidos para outras linguagens de programação, buscando utilizar maneiras de otimizar e melhorar a qualidade dos testes. Como exemplo, pode-se citar os trabalhos de McKeeman [McKeeman 1998] que utilizou esta abordagem para testes na linguagem C, e Ofenbeck et al. [Ofenbeck et al. 2016] que fez uso desta abordagem para testar o compilador de C++, dentre outros.

3. Teste Diferencial de Compiladores

O teste diferencial de compiladores, tem como foco principal a utilização de casos de teste (que podem ser gerados aleatoriamente) para comparar dois ou mais sistemas [McKeeman 1998]. Os compiladores devem passar por uma bateria de testes, onde ambos devem receber os mesmos parâmetros de entrada, e caso ocorra uma incongruência nos testes, como um *crash* ou um *loop* por exemplo, isto indicará que existe um *bug* em um dos compiladores, tendo como probabilidade maior um erro no compilador não oficial [Ofenbeck et al. 2016].

O maior desafio no teste diferencial é garantir a qualidade dos testes, já que é necessário que parâmetros eficientes sejam utilizados. Neste sentido, a utilização de testes gerados de forma aleatória podem auxiliar no processo, uma vez que tendem a ter uma cobertura maior do código base do compilador. Ainda assim, é possível que sejam relatados falsos positivos, uma vez que, dependendo da situação, mesmo com resultados diferentes entre as execuções, ambas podem estar corretas.

4. Geração de Código Aleatório

O mecanismo de geração de código Java aleatório já desenvolvido [Kraus et al. 2021] utiliza uma série de bibliotecas, que permitem deste a extração de informações de classes pré-existentes usando *Java Reflection*[1], a composição de ASTs e compilação para código Java através do *JavaParser*[2], e a especificação de testes baseados em propriedades com o framework *Jqwik*[3].

A geração do código é orientada por um tipo válido, ou seja, primeiro são extraídas informações sobre classes e interfaces, e definido de forma aleatória um tipo primitivo ou definido pelo usuário, para que seja produzida uma expressão ou diretiva seguindo esta definição. Recursivamente são gerados acessos a atributos públicos, invocação de métodos, construção de objetos, diretivas condicionais ou de repetição, etc. Para apresentar parte do mecanismo de geração, abaixo é apresentado um trecho de código que demonstra a geração de diretivas em termos gerais (condicionais, repetição e expressões), e um método que apresenta a geração de diretivas condicionais.

```
1  @Provide
2  public Arbitrary<Statement> genStatement() {
3     //Gerar possiveis Statements
4     return Arbitraries.oneOf(genIfStmt(), genWhileStmt(), genExpressionStmt));
5  }
6  @Provide
7  public Arbitrary<IfStmt> genIfStmt() {
8      Arbitrary<Expression> e = mCore.genExpression(PrimitiveType.booleanType());
9      return e.map(exp -> new IfStmt(exp, genStatement().sample(), genStatement().sample()
           ));
10 }
```

Como pôde ser percebido no código acima, o método *genStatement* realiza a seleção de forma aleatória de uma possível diretiva válida. No método *genIfStmt* é gerada uma diretiva condicional, onde primeiramente é gerada uma expressão com o tipo *booleano* através da chamada do método *genExpression*, para então combinar com duas novas diretivas (*genStatement*) para os casos verdadeiros e falsos da expressão condicional. Esse mecanismo de geração será adaptado para produzir entradas válidas para a ferramenta de teste diferencial de compiladores Java, a qual está sendo desenvolvida neste projeto.

5. Resultados Parciais

Este é um projeto que encontra-se em desenvolvimento, sendo que parte importante deste já foi desenvolvida [Kraus et al. 2021]. A geração de programas para servir de entrada para os casos de teste está concluída, podendo sofrer pequenas adaptações para a aplicação no contexto descrito neste texto. A Figura 1 apresenta o fluxo de execução da ferramenta para teste diferencial de compiladores que está sendo desenvolvida.

O fluxo de execução inicia com o recebimento de uma série de classes / interfaces Java (*imports*), dos quais serão extraídas as informações sobre os tipos definidos pelo usuário, como os nomes de classes ou interfaces, seus atributos, métodos e construtores, etc. Estas informações servem como base para o mecanismo de geração de código aleatório, o qual é responsável por produzir milhares de programas diferentes, os quais

[1] https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/package-summary.html

[2] www.javaparser.org

[3] www.jqwik.com

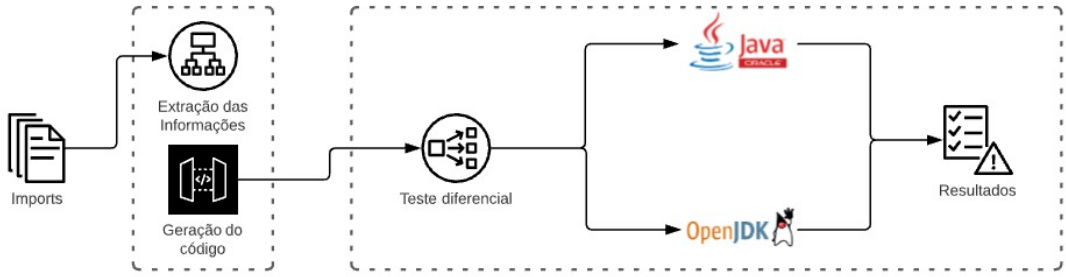

Figura 1. Fluxo de execução proposto.

servirão como entrada (casos de teste) para o instrumento de teste diferencial de compiladores.

Na etapa final, que ainda deve ser desenvolvida, os casos de teste serão compilados por diferentes compiladores e executados individualmente nas suas respectivas máquinas virtuais. Estatísticas de compilação e execução (comportamento) de ambos os compiladores/JVMs serão coletadas e comparadas para verificar a existência de inconsistências, o que será um indicativo de possíveis erros nas ferramentas em estudo. Por exemplo, diversos programas gerados aleatoriamente serão compilados em diferentes implementações da linguagem Java, verificando a propriedade de compilação. De forma similar, os mesmos programas serão executados nas diferentes implementações, e os resultados (estado e output) serão comparados, verificando assim a propriedade de comportamento. Em ambas as verificações, caso existam divergências entre as ferramentas testadas, um indicativo de erro será reportado. A partir da análise dos erros detectados pela ferramenta, serão abertos relatórios de *bug* para os respectivos compiladores. Como referência, o compilador de Java da Oracle será utilizado como implementação oficial para os testes de outros compiladores.

6. Resultados Esperados e Contribuições

Este projeto consiste no estudo de teste diferencial de compiladores, mais especificamente aqueles referentes à linguagem Java, através da execução de casos de teste gerados de forma aleatória. A hipótese levantada para este trabalho é que através da utilização destes casos de teste é possível ter uma maior cobertura dos casos implementados nos compiladores, permitindo a detecção de *bugs* antes da liberação destes sistemas para os usuários finais (programadores), evitando a introdução de erros em cascata em projetos de software.

Esta proposta pretende contribuir para o avanço desta área de pesquisa, fornecendo uma ferramenta de software livre para a execução de teste diferencial de compiladores Java através de código gerado aleatoriamente. Além disso, o mecanismo de geração de código e a metodologia proposta neste projeto também podem ser aproveitadas para outros escopos ou outras linguagens de programação.

Referências

Cass, S. (2019). As principais linguagens de programação de 2019. IEEE Spec.

Claessen, K. and Hughes, J. (2000). Quickcheck: A lightweight tool for random testing of haskell programs. *SIGPLAN Not.*, 35(9):268–279.

Kraus, L. F., Schafaschek, B., Ribeiro, R. G., and da Silva Feitosa, S. (2021). Synthesis of random real-world java programs from preexisting libraries. In *25th Brazilian Symposium on Programming Languages*, SBLP'21, page 108–115, New York, NY, USA. Association for Computing Machinery.

McKeeman, W. M. (1998). Differential testing for software. *DIGITAL TECHNICAL JOURNAL*, 10(1):100–107.

Ofenbeck, G., Rompf, T., and Püschel, M. (2016). Randir: Differential testing for embedded compilers. In *Proceedings of the 2016 7th ACM SIGPLAN Symposium on Scala*, SCALA 2016, page 21–30, New York, NY, USA. Association for Computing Machinery.

Palka, M. H., Claessen, K., Russo, A., and Hughes, J. (2011). Testing an optimising compiler by generating random lambda terms. In *Proceedings of the 6th International Workshop on Automation of Software Test*, AST '11, pages 91–97, New York, NY, USA. ACM.