A Simplified Method for Automatic Verification of Java Programs

1st Zhi Li
School of Computer Science and Engineering
Guangxi Normal University
Guilin, China
zhili@gxnu.edu.cn

2nd Ling Xie
School of Computer Science and Engineering
Guangxi Normal University
Guilin, China
lingxiehy@outlook.com

3rd Yilong Yang*
School of Software
Beihang University
Beijing, China
yilongyang@buaa.edu.cn

***Abstract*—Current KeY verification tool for Java programs provides limited capability for verifying Java programs. In order to solve this problem, we provide a method for simplifying complex Java programs into a format that is compatible with the KeY. A set of simplification rules based on abstract syntax tree (AST) are proposed. These rules can keep the logic and semantics of the original Java programs mostly unchanged, while meeting the requirements of KeY verification tool. The paper concludes with a bank ATM example to demonstrate the feasibility of our work.**

***Index Terms*—Program verification, KeY verification tool, abstract syntax tree (AST), Java program simplification**

I. Introduction

Program verification is an important part of software development, which can detect some errors in the program. Current program verification methods or techniques are only applicable to program fragments or simple programs. Therefore, a program simplification method is urgently needed to extend the capability of existing program verifiers so that complex programs can be dealt with constructively. In this paper, we provide a simplification method to enable the KeY tool[4]to verify those Java programs with moderate complexities.

The Java simplification work in this paper uses the abstract syntax tree (AST) to parse, traverse, and re-factor code. Inspired by the work of [1,2,3], AST is used to parse the variables, types and functions of the source code and then traverse and re-factor the object code, thus simplifying the Java program while keeping most of the logic and semantics unchanged.

The ultimate goal of simplifying complex Java programs is to be verifiable in the KeY tool. The KeY is a formal program tool for Java programs, with both fully automated and interactive verification. It converts the Java program as input to Java dynamic logic (JavaDL), and then verifies the JavaDL step by step applying the corresponding taclet. Finally, the verification results are presented in the form of a proof tree[4]. As a continuously improved tool, KeY supports and verifies invariant specifications. The specification and verification of invariant allows us to conveniently specify and verify strong data integrity properties for Solidity smart contracts[5].

*corresponding author: yilongyang@buaa.edu.cn
DOI reference number:10.18293/SEKE2022-180

II. Method and Implementation

A. Overview of the methods

To enable the KeY tool to verify complex Java programs, this paper presents 7 simplified rules.Details are as follows:

1) New AST Rule

$$New\ AST\ Rule\ \frac{Content}{CompilationUnit}$$

Take the source program (Content) as a parameter of the *createAST* method to generate an AST CompilationUnit.

2) Get Type Rule

$$Get\ Type\ Rule\ \frac{CompilationUnit}{TypeDeclaration}$$

With the AST as the head node, the children node is called step by step. The type of this rule includes the class name and the content of all methods, and is the header of all modification nodes.

3) Modify Data Type Rule

$$Modify\ Data\ Type\ Rule\ \frac{float}{int}$$

Modify the data type after the node location is found, then the method is called to replace the previous data type with the new data type.

4) Delete Rule

$$Delete\ Rule\ \frac{Object}{\phi}$$

The Delete rule includes deleting comments, deleting variables, deleting statements, etc. The node where the deleted object is located in the AST is found, and the *remove* or *delete* method is called to delete it.

5) Substitution Rule

$$Substitution\ Rule\ \frac{getPasswordValidated()}{passwordValidated}$$

Substitution rule here refers to the get and set methods that replace new variables. The node that calls the *get* and *set* methods in AST is found, and new variables are generated by creating new methods, and a function is called to replace the

left and right sides of the get and set method expressions, and the value of the operator is kept unchanged.

6) Add Rule

$$Add\ Rule\ \frac{\phi}{Object}$$

Add Rule finds the node position of the variable or method call to be added, and after generating a new expression, insert it into the corresponding node position to complete the operation of adding.

7) Simplify For Loop Rule

$$Simplify\ For\ Loop\ Rule\ \frac{EnhancedForStatement}{ForStatement}$$

This rule refers to replacing the enhanced *for* loop with a generic *for* loop. Find the enhancement *for* loop and delete it, create a general *for* loop, and insert the new *for* loop into the node location where the original enhancement *loop* is located.

B. Method implementation

The implementation of simplified methods is based on the eclipse *JDT plug-in*, the specific method implementation is divided into the following steps:

1) Environment preparation: Since you are using the *JDT plug-in*, the first step is to install the *JDT plug-in* in Eclipse,the second is to configure an environment suitable for the *org.eclipse.jdt.core.dom** class, that is, to download the corresponding JAR package to use when the configuration program runs.

2) Parsing the AST: Firstly, use *ASTParser parsert = ASTParser.newParser(AST. JLS3)* statement to create a parser. Secondly, the Java source program to be parsed is generated as a string-typed parameter of the parser source code in AST. Finally, use the parser to create and return the AST context result CompilationUnit as the root node.

3) Modifying the AST:

a.Modify the data type: modify float into int.

b.Delete function implementation:delete variables, comments, statements,etc.

c.Substitution function implementation: Variable substitution *get* or *set* method.

d.Add function implementation: Add a variable or method call.

e.Simplified for loops: Rewrite the enhanced *for* loop as a general *for* loop.

4) AST convert into Java program: The file output stream (FileOutputStream) parses the modified string, and finally converts the string into a Java file and outputs it to the specified location.

III. EXPERIMENT

This section presents an example of a bank ATM withdrawal that shows how to simplify the source Java program by using AST and then verify the correctness with the KeY tool. Figure at github(https://github.com/1713022804/ATMexample) shows the simplification process of the bank ATM withdrawal, represented in AST, in which those on the left of the dashed line represents the original complex Java program, while the right side represents the simplified Java program.

Determine the format that Java programs verify in KeY, and then make specific simplifications according to the 7 rules provided in Section A of II. In this example, we mainly simplify the seven functions of the source Java program. The following are two examples showing how our rules are applied:

(1)In the *depositFunds* function, the Modify Data Type Rule is used to modify the parameter Float type into the Int type. The Delete Rule is applied to remove unnecessary comments and variables for verification of the method. Then the Substitution Rule is used to the *getPasswordValidated()* method, which is replaced by the PasswordValidated variable.

(2)In the *inputCard* function, the enhanced *For* Loop is simplified into the general *For* Loop by using the Simplify For Loop Rule, and Add Rule is applied to add variable C of BankCard data type to the method, then the Delete Rule is used to Delete comments, variables or statements in the methods. Finally, The *GetCardIDValidated()* method is replaced by the CardIDValidated variable, following the Substitution Rule.

IV. CONCLUSIONS

This paper presents seven rules you can use when simplifying your source Java programs into a format that the KeY can verify. Based on the AST, the simplified rules are derived, which are illustrated by using corresponding examples, and the simplified Java programs are verified based on the Java program-oriented verification tool KeY. In the future, we will continue to improve the simplification work based on AST and try to use empirical methods to evaluate our simplification rules.

ACKNOWLEDGMENT

This work is partially supported by the National Natural Science Foundation of China (61862009).

REFERENCES

[1] S. Horwitz. Identifying the semantic and textual differences between two versions of a program. In Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), pages 234–245, June 1990.

[2] G. Stoyle, M. Hicks, G. Bierman, P. Sewell, and I. Neamtiu. Mutatis Mutandis: Safe and flexible dynamic software updating. In Proceedings of the ACM SIGPLAN/SIGACT Conference on Principles of Programming Languages (POPL), pages 183–194, January 2005.

[3] Neamtiu I, Foster J S, Hicks M. Understanding source code evolution using abstract syntax tree matching[C]//Proceedings of the 2005 international workshop on Mining software repositories. 2005: 1-5.

[4] Ahrendt W, Beckert B, Bubel R, et al. Deductive Software Verification-The KeY Book[J]. Lecture notes in computer science, 2016, 10001.

[5] Ahrendt W, Bubel R. Functional verification of smart contracts via strong data integrity[C]//International Symposium on Leveraging Applications of Formal Methods. Springer, Cham, 2020: 9-24.