Develop and Evaluate a Security Analyser for Finding Vulnerabilities in Java programs

A dissertation submitted to The University of Manchester for the degree of Master of Science in the Faculty of Science and Engineering

2021

By

Tong Wu (10772895)

Department of Computer Science

Contents

Word Count: 12982

Abstract

The purpose of this dissertation is to understand the use of a software verification technique for Java to discover security vulnerabilities in Java programs, and implement appropriate extensions to validate the results produced by the Java software verifier. In this research, the software verifier refers to the Java Bounded Model Checker (JBMC), which is a BMC-based checker for Java bytecode. It can output the verification results to a GraphML format file called witness, which can indicate the path to the specific attribute state in the program. However, we cannot be sure whether the witnesses generated by JBMC for all tasks can correctly indicate the path to the detected attribute state. Therefore, validating the witnesses is necessary for increasing the trust level of JBMC.

In the annual Competition on Software Verification (SV-COMP) held at TACAS, people have not yet provided a witness validation tool for Java programs, and the witness validation for Java is still in the beginning stage. This dissertation introduces the use of JBMC to benchmark the Java benchmarks in the SV-COMP, and develop a new tool to validate the witnesses generated by JBMC. At the same time, effective and reliable management of resources is realized when benchmarking JBMC and the witness validator. In this dissertation, we focus on using Python scripts to run witness validation, and integrate the validation tool into BenchExec ecosystem so that it can implement resource management at runtime and contribute to the SV-COMP. The following chapters will describe the extension and its evaluation and further work.

Declaration

No portion of the work referred to in the dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

I would like to thank the university for providing me with high-quality learning resources, especially my supervisor, Dr Lucas Cordeiro, for guiding my research direction and making progress during the entire project research process. At the same time, I would also like to thank my family and friends for their care for me. I could not have done it without all this support.

1. Introduction

1.1 Problem description

With the development of the Internet and the popularity of the Java programming language, more and more complex Java codes are being built. According to PYPL statistics, as of May 2021, Java is the second most popular language in the world [1].

Minor security incidents may cause high recovery costs. Computerworld UK staff listed the ten most serious software failures in recent history. For example, British Airways' IT failure caused hundreds of flights to be cancelled or delayed [2]. Thus, it is very important to fully conduct various tests at the beginning of a software project cycle to avoid catastrophic software security problems.

Since errors and defects that are rarely reached in the program are difficult to be detected completely through traditional simulation and testing methods, people introduced the model checking technique as a supplement, which can automatically check whether the software being built has errors by checking all the achievable states [3]. In this project, we focus on JBMC, a bounded model checking tool for Java, to detect vulnerabilities in the Java programs [4]. When JBMC detects a program vulnerability, it generates a counterexample. However, JBMC is not yet fully mature, and it sometimes produces incorrect verification results, such as a false alarm [5]. Therefore, it is necessary to implement appropriate extensions to validate the counterexamples generated by the verifier, thereby enhancing the trustworthiness of JBMC.

Vi implements an extension to validate the witnesses generated by JBMC by creating and executing Java unit tests. but due to the limitations of its validation algorithm, it only supports validating some simple Java benchmarks in the SV-COMP. In addition, since the content of Java benchmarks must be modified before running, it cannot be directly used as a validation tool for the SV-COMP to benchmark the Java benchmarks [6]. This drives the need to develop a better performance validation tool that can be used as an executable tool in the SV-COMP and supports as many Java benchmarks in the SV-COMP as possible.

1.2 Aims and objectives

In this paper, the aims and objectives are as follows.

Aims:

- Evaluate the existing verification strategies in JBMC to find security vulnerabilities.
- Implement suitable extensions for the verifier to verify large programs.

Objectives:

- Gain a good understanding of JBMC to understand the strategy used to detect security vulnerabilities in Java programs.
- Implement a witness validation tool for Java to solve the existing limitations to have a better performance over the benchmarks in the SV-COMP.
- Integrate the validation tool into BenchExec ecosystem to have precise resource limits and measurement that can contribute to the SV-COMP.
- Evaluate the performance of the witness validator.
- Evaluate the soundness and completeness of JBMC.

1.3 Contribution

This research project produces two main implementations which is a witness validation script and a startup script. Both of the scripts are written in Python language. The witness validation script reads the violation witness file produced by JBMC to extract counterexamples from it, and then assigns the counterexamples to the newly generated Java programs and runs them with JVM to reproduce the identified error which is found by JBMC. The startup script benchmarks JBMC and the witness validator in BenchExec, and finally generates an interactive table to display all the results.

BenchExec is a benchmarking tool that can execute commands with a large set of input files [7]. In this project, it is used with the aim of benchmarking and resource limitation.

1.4 Organization of dissertation

The structure of this dissertation starts from Chapter 2, discussing relevant background knowledge. The following Chapter 3 introduces the implementation of the extensions in detail. Then, in Chapter 4, the experimental results are presented and analysed. Finally, Chapter 5 summarizes the limits of this extension and the work worth continuing to do in the future.

2. Background

This chapter will discuss some of the relevant background of the dissertation, so that readers can fully understand the project. There are six subchapters in total. First, subchapter 2.1 introduces security vulnerabilities and their threats to the industry. Then, subchapter 2.2 introduces software testing and its significance in reducing program vulnerabilities. Next, subchapter 2.3 introduces the bounded model checking technique and its instantiation tool JBMC, which is the Java verifier used in our project. After that, subchapter 2.4 discusses witness validation, which lists some existing witness validation tools, and discusses the limitations of an existing witness validator for Java in detail. Finally, subchapter 2.5 introduces the BenchExec framework which is applied to the project and subchapter 2.6 summaries the main findings.

2.1 Security vulnerabilities

The Internet of Things technology (IoT) plays an increasingly important role in people's communication systems. Research shows that by 2020, 30.7 billion devices are connected to the Internet of Things, and this number will reach 75.4 billion in 2025[8]. As this network becomes more and more complex, security issues have become a serious challenge. The Internet of Things (IoT) architecture includes three basic layers: perception layer, network layer and application layer, and different layers may be vulnerable to vulnerabilities[9].

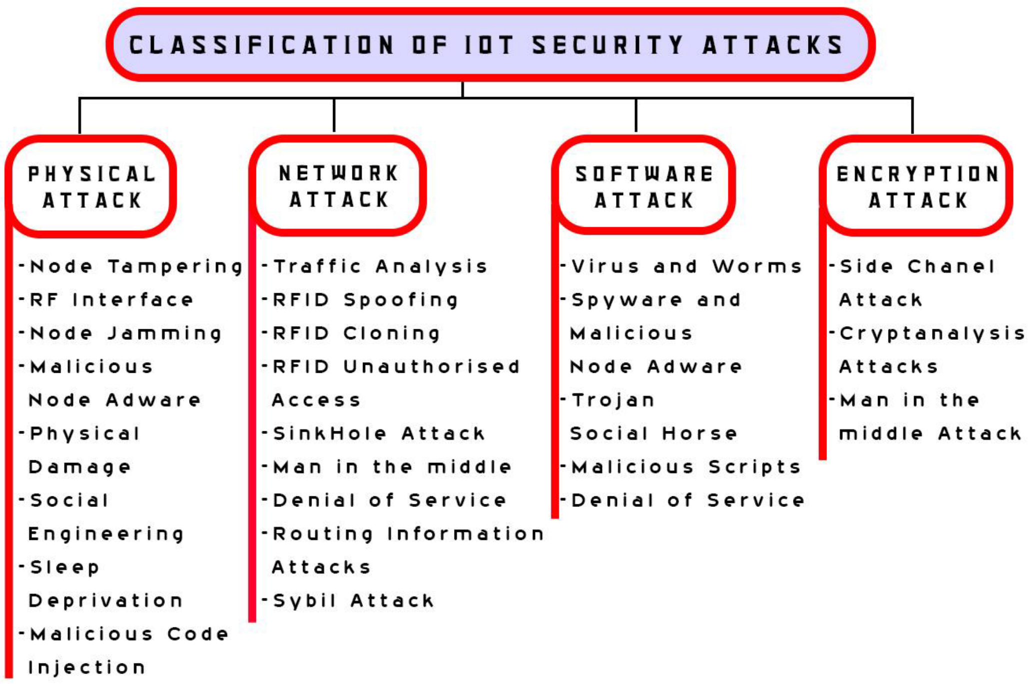


Figure 2.1: Classification of IoT Security Attacks

Source [10]

The vulnerabilities of IoT can exist on devices, data, software, and networks. As shown in Figure 2.1, Intruders can use various methods to implement physical attacks, network attacks, software attacks, and encryption attacks.

Software security includes building programs that can still function normally under malware attacks, and vulnerability is the root cause of malware attack. In the SonicWall report, the number of global IoT malware attacks in 2018 increased by 215.7% year-on-year [11]. Since the Internet of Things is facing many security challenges, we need to conduct more research on it and design reliable algorithms and tools to detect and repair various possible vulnerabilities in the Internet of Things and improve the security of the Internet of Things.

```
SELECT * FROM users WHERE username = 'administrator'--' AND password = ''
```

Figure 2.1.2: Example of SQL injection

Source [12]

Although Java runs on the server side and is considered a language with good security, hackers still have many ways to attack it to steal users' privacy and expose users to security threats, such as SQL injections [13]. There are many ways to illegally obtain database information by implementing SQL injections to attack the system. Figure 2.1.2 shows an example to skip password verification to log in to the system through SQL injection. When the Java program obtains this string to execute the database command, the content behind the `administrator` will be used as the SQL comment, so the password verification process is skipped and the login is illegal.

2.2 Software testing

Testing plays a very important part in confirming that the software has met the functional and non-functional requirements in the software development process. Software testing includes defining test conditions, designing test cases, and executing test cases. This is followed by analysis and reporting the test results. In other words, software testing is the review or comparison process between actual output and expected output. It can increase confidence before releasing the product to potential customers, and the recommendations made by the testing department based on the analysis of the test results play a key role in deciding whether to release the product [14].

Even the most well-known companies occasionally deliver software with quality problems. Defects in this software may cause different levels of loss to customers, and may be life-threatening in serious cases [14]. But this does not mean that the company's attitude in the software development process is not serious or irresponsible, but because with the continuous expansion of the software scale and the complexity of the program, it is difficult for people to fully consider all potential defects. When we eliminate a defect, it may make the program more complicated, resulting in more potential defects [15].
Therefore, it is necessary to conduct multiple tests as much as possible to improve the confidence of the software.

Software testing methods can be divided into two types:

- Black box testing

Black box testing is usually a functional test, which aims to validate whether we are building the right software. In black box testing, the tester does not know the internal structure of the source code, so the code looks like a "black box" to him. The tester enters the test case according to the software requirements, obtains the returned result and judges whether it is the result he expects [16].

- White box testing

White box testing is usually structural testing. It is designed to verify whether the software we build is right. In white box testing, the tester is usually the person who understands the specific structure of the source code, such as the developer of the code [16].

There are different stages of software testing:

- Unit Testing

Unit testing is the first stage of software testing. When developers have completed the source code, they need to design, write and execute test cases before submitting to the testers to cover as much as possible in each branch of the code. This is also a white box test. Unit testing can find and eliminate program defects at the earliest and reduce the cost of repairing code [17].

- Integration testing

Integration testing is to adopt appropriate measures to integrate the units that have completed the unit test to test whether the integrated functions are correct. The test can be performed by developers or testers, usually to check whether the interfaces between the functional units are correct [17]. It can belong to both white box testing and black box testing, depending on whether the tester is its developer or an independent tester.

- System testing

System testing is usually executed by a separate testing team. It is to use the software to be tested as an element of the computer system, to test whether the software meets the software requirements when running in the system, and to test its support by combining computer software and hardware and other elements [17].

- Regression testing

Regression testing is also an important part of the software life cycle. It is designed to check whether the modified code remains intact, so multiple regression tests should be performed at all stages of software development [17].

As a method to reduce software and system security vulnerabilities, software testing needs to be implemented many times in each cycle of software development.

2.3 Bounded model checking

Since it is impossible to easily find all the defects of a complex software by designing test cases, model checking tools are introduced as a supplement to software testing. Because compared with traditional software testing which can only design limited test cases, model checking tools can consider all possible behaviours of a finite system [3].

2.3.1 Model checking

Model checking is a verification technique that can automatically check the temporal properties of a finite system. This concept was created by Clarke and Emerson in the 1980s. When the state attribute of the system fails, a counterexample will be produced. This counterexample is the trace of system states, where the last state violates the attribute. The first checking algorithm was to enumerate the reachable states of a finite system, but for a system with an exponentially increasing number of states, the checking ability is very limited [3].

2.3.2 Symbolic model checking

Based on the limitations of early model checking on the system of exponentially increasing state sets, symbolic model checking technique was introduced. In this technique, a certain behaviour of the system is determined by some variables whose values are 0 or 1, then the state list of the system is represented by some Boolean functions. And the Boolean formulas can be effectively processed using Reduced Ordered

Binary Decision Diagrams (BDD), which is a diagram representation of Boolean functions. This technique realizes the verification of the system in reality and is gradually adopted by the industry. As the number of Boolean functions representing the state set increases exponentially, storing and operating BDD will also require an increasing amount of memory [3].

2.3.3 Bounded model checking

Bounded model checking (BMC) technique uses SAT solvers to solve Boolean formulas instead of BDD, so it does not have the memory problem of BDD-based model checkers. Its basic idea is to find counterexamples in a given execution depth, which is bounded to a given bound k. If the checker does not find a counterexample in the finite states within the bound k, it cannot prove that all the states of the system are satisfiable, because the state at the bound k+1 may not be satisfied [3].

JBMC is an open-source verifier that based on the bounded model checking technique for Java Bytecode. It is an extension to the C Bounded Model Checker (CBMC) which is developed on top of the CProver framework[4]. It won the third place in the verification results of the Java benchmarks in the 10th Competition on Software Verification, where it successfully verified 180 correct results and found 243 program errors. However, due to its limits, there are still 50 wrong or unknown verification results [18].

Run the following command line in the terminal to execute JBMC:

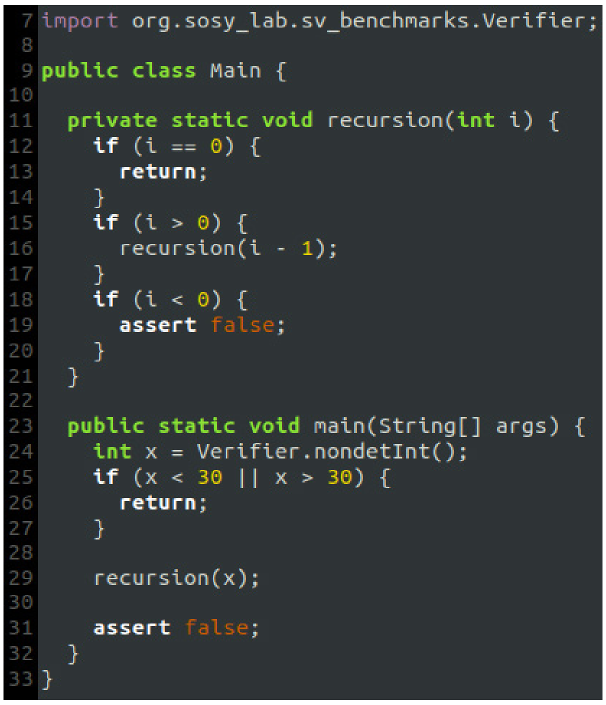
```
jbmc <file> <option>
```

It does not accept Java source code as input, but only accepts class file or jar files as input. Besides, it can also allow further specification of other properties. For example, whether to generate a witness file, the bound k to be checked, etc.

The following is a command example to run JBMC. The name of the input class file is `Main`. In the following options, `-unwind 100` means to unwind the loop in the program 100 times, and `--graphml-witness file` means to generate a witness file named `file`.

```
jbmc Main -unwind 100 --graphml-witness file
```

The difference in the times of loop unrolling in the program also affects the verification result.

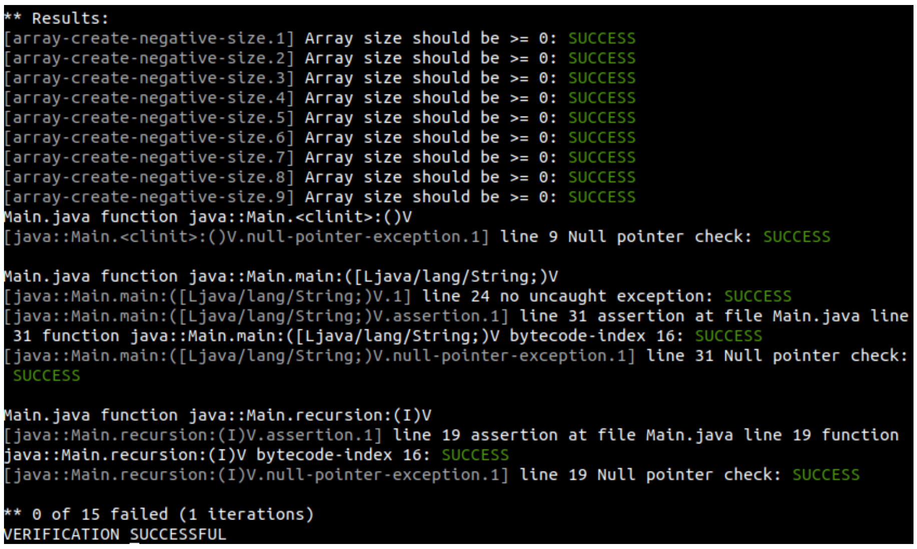


```
 7 import org.sosy_lab.sv_benchmarks.Verifier;
 8
 9 public class Main {
10
11   private static void recursion(int i) {
12     if (i == 0) {
13       return;
14     }
15     if (i > 0) {
16       recursion(i - 1);
17     }
18     if (i < 0) {
19       assert false;
20     }
21   }
22
23   public static void main(String[] args) {
24     int x = Verifier.nondetInt();
25     if (x < 30 || x > 30) {
26       return;
27     }
28
29     recursion(x);
30
31     assert false;
32   }
33 }
```

Figure 2.2: A recursive program example

In this code in Figure 2.2, x is an int variable with random value. When the value of x is 30, the recursive function in the program will be recursively called 30 times, and finally the value of `i` is reduced to 0, reaching the assertion error on line 31 of the program. Therefore, according to the BMC theory, if this recursive function is unrolled within 30 times, no assertion error will be detected.

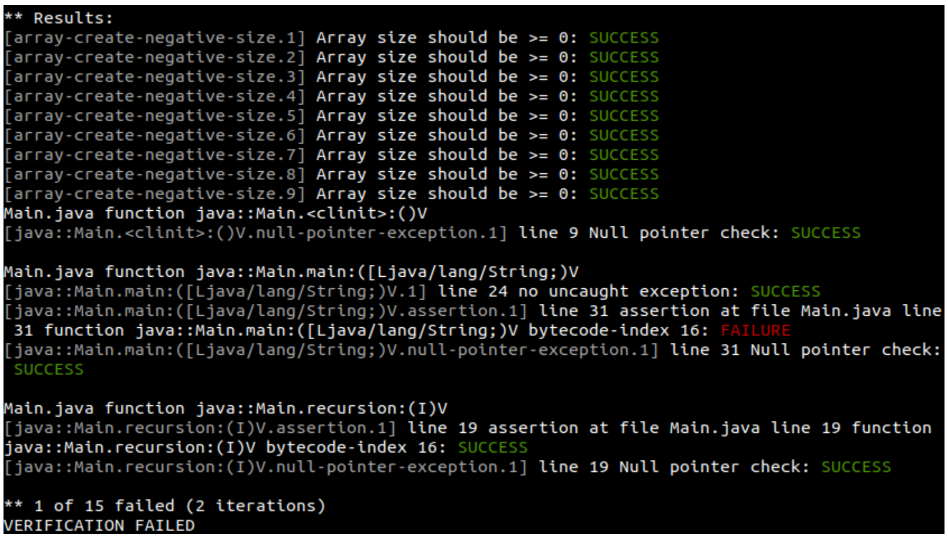
```
jbmc Main -unwind 10
```



```
** Results:
[array-create-negative-size.1] Array size should be >= 0: SUCCESS
[array-create-negative-size.2] Array size should be >= 0: SUCCESS
[array-create-negative-size.3] Array size should be >= 0: SUCCESS
[array-create-negative-size.4] Array size should be >= 0: SUCCESS
[array-create-negative-size.5] Array size should be >= 0: SUCCESS
[array-create-negative-size.6] Array size should be >= 0: SUCCESS
[array-create-negative-size.7] Array size should be >= 0: SUCCESS
[array-create-negative-size.8] Array size should be >= 0: SUCCESS
[array-create-negative-size.9] Array size should be >= 0: SUCCESS
Main.java function java::Main.<clinit>:()V
[java::Main.<clinit>:()V.null-pointer-exception.1] line 9 Null pointer check: SUCCESS

Main.java function java::Main.main:([Ljava/lang/String;)V
[java::Main.main:([Ljava/lang/String;)V.1] line 24 no uncaught exception: SUCCESS
[java::Main.main:([Ljava/lang/String;)V.assertion.1] line 31 assertion at file Main.java line
 31 function java::Main.main:([Ljava/lang/String;)V bytecode-index 16: SUCCESS
[java::Main.main:([Ljava/lang/String;)V.null-pointer-exception.1] line 31 Null pointer check:
 SUCCESS

Main.java function java::Main.recursion:(I)V
[java::Main.recursion:(I)V.assertion.1] line 19 assertion at file Main.java line 19 function
java::Main.recursion:(I)V bytecode-index 16: SUCCESS
[java::Main.recursion:(I)V.null-pointer-exception.1] line 19 Null pointer check: SUCCESS

** 0 of 15 failed (1 iterations)
VERIFICATION SUCCESSFUL
```

Figure 2.3: Unwind the program 10 times

```
jbmc Main -unwind 30
```



```
** Results:
[array-create-negative-size.1] Array size should be >= 0: SUCCESS
[array-create-negative-size.2] Array size should be >= 0: SUCCESS
[array-create-negative-size.3] Array size should be >= 0: SUCCESS
[array-create-negative-size.4] Array size should be >= 0: SUCCESS
[array-create-negative-size.5] Array size should be >= 0: SUCCESS
[array-create-negative-size.6] Array size should be >= 0: SUCCESS
[array-create-negative-size.7] Array size should be >= 0: SUCCESS
[array-create-negative-size.8] Array size should be >= 0: SUCCESS
[array-create-negative-size.9] Array size should be >= 0: SUCCESS
Main.java function java::Main.<clinit>:()V
[java::Main.<clinit>:()V.null-pointer-exception.1] line 9 Null pointer check: SUCCESS

Main.java function java::Main.main:([Ljava/lang/String;)V
[java::Main.main:([Ljava/lang/String;)V.1] line 24 no uncaught exception: SUCCESS
[java::Main.main:([Ljava/lang/String;)V.assertion.1] line 31 assertion at file Main.java line
 31 function java::Main.main:([Ljava/lang/String;)V bytecode-index 16: FAILURE
[java::Main.main:([Ljava/lang/String;)V.null-pointer-exception.1] line 31 Null pointer check:
 SUCCESS

Main.java function java::Main.recursion:(I)V
[java::Main.recursion:(I)V.assertion.1] line 19 assertion at file Main.java line 19 function
java::Main.recursion:(I)V bytecode-index 16: SUCCESS
[java::Main.recursion:(I)V.null-pointer-exception.1] line 19 Null pointer check: SUCCESS

** 1 of 15 failed (2 iterations)
VERIFICATION FAILED
```

Figure 2.4: Unwind the program 30 times

The two figures above are the verification results of unfolding the recursive function of the program 10 and 30 times by JBMC respectively. It can be seen that no program assertion error was found when the recursive function was unfolded 10 times, and the program assertion error was detected when the recursion function was unfolded 30 times.

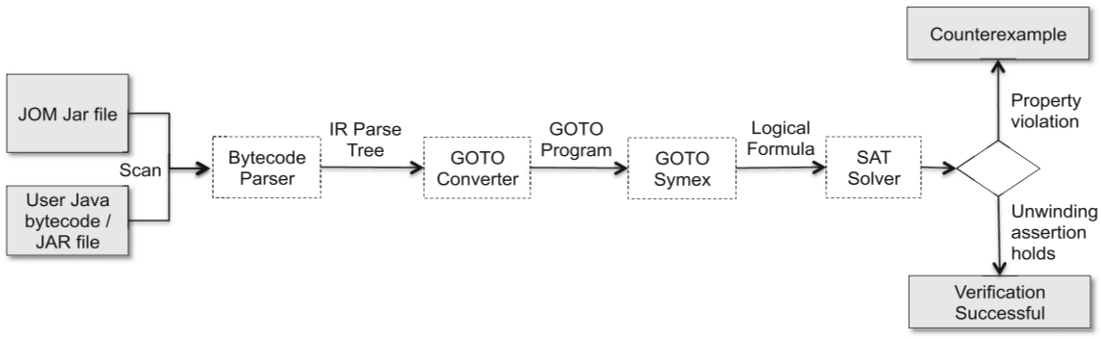


Figure 2.5: JBMC Architecture

Source [19]

Figure 2.5 illustrates the JBMC architecture, which consists of three main parts. The grey rectangle on the left side of the chart represents the input. The white rectangle in the middle represents the verification steps of JBMC. The grey rectangle on the right represents the output verification result.

First, JBMC receives a JAR file or class file as input. Then, it uses a bytecode parser to parse the input file into a parse tree. Next, it uses the GOTO converter to convert the parse tree into a GOTO program. After that, it uses the GOTO Symex to translate the GOTO program to Boolean logic formulas. Finally, the Boolean logic formula will be checked by a SAT Solver to determine if the formula is satisfiable. If no UNSAT results are returned by the SAT Solver, which means all property holds in the program, then JBMC will output "Verification Successful". Otherwise, if a UNSAT result is returned by the SAT Solver, which means there exists a violation property, then JBMC will output "Verification Failed" and also produced a counterexample.

Although JBMC has good verification performance, it still has some areas for improvement. First of all, due to the feature of bounded model checking, it cannot check the state upper the bound k. Second, it has some restrictions on String object operations, such as regular expressions. The witness he produced on this is also difficult to validate. Third, it can be expanded to support verification of more Java libraries. Finally, it cannot effectively support multi-threaded program verification [19].

In addition to JBMC, there are some other Java verifiers. Such as JPF [20] and JayHorn [21]. JPF is an explicit-state model checker that is used to find defects along all potential paths [20]. JayHorn is a Java verifier that can produce Horn clauses to encode the verification condition [21].

2.4 Witness validation

2.4.1 Witness

The trace to the specific property within the finite bound k is called the witness for the property. But for a finite path without a back loop, even if all the states are satisfied, it does not mean that it has found a witness [3].

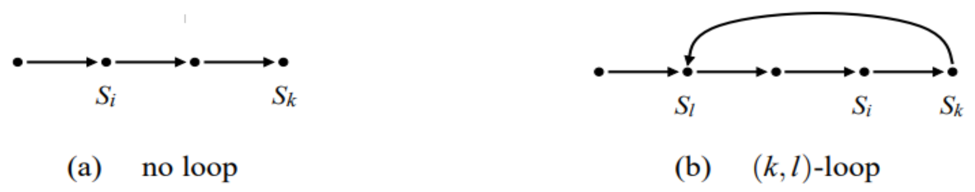


Figure 2.6: The two cases for a bounded path

Source [3]

The Figure 2.6 shows two cases for a bounded path. The difference between the two paths in the figure is that there is a back loop in the finite path on the right, so it is still an infinite path. For the finite path on the left of the figure, it cannot reach the infinite path outside the bound k, so it is unable to prove whether the states outside the bound k is satisfied [3].

Witnesses can guide us the path to the expected property. A witness can be one of the two forms: a correctness-witness or a violation-witness. On one hand, a correctness-witness can guide us the path to show why the formula is satisfiable over the model. On the other hand, a violation-witness gives counterexamples for us to find the violation property to prove why the formula is not satisfiable over the model [22].

The witness produced by a model checker, such as JBMC, are represented by the XML-based format GraphML. However, the witness cannot guarantee to be a valid one since the verifier may produce false results, and the format of the counterexamples are complicated and difficult to read by subsequent programs, which pushes the need to do the witness validation.

2.4.2 Witness validation for C

The categories in SV-COMP provide guidance for which verifier is suitable for which kinds of programs, but the verification technique cannot be fully applied in practice because it has an important unsolved problem: the verifier may generate false alarms, and validating the witnesses generated by the verifier may take a lot of effort. Witness validation is a process of checking whether the same results can be reproduced independently according to the given programs, specifications, verification results and the generated witnesses, thereby improving the trust level of the model checking tool [5].

However, the development of witness validation technique is still at an early stage. At present, there are seven open-source witness validators participated in the SV-COMP, but all of them are for C programs. Here we introduce the MetaVal and the NitWit.

MetaVal: The concept of the C witness validator MetaVal (see Figure 2.7) is to generate a new program based on the input source program and witness, and then use any available software verifier to check if the new program contains a specific assertion error, or if all the assertions hold[23].

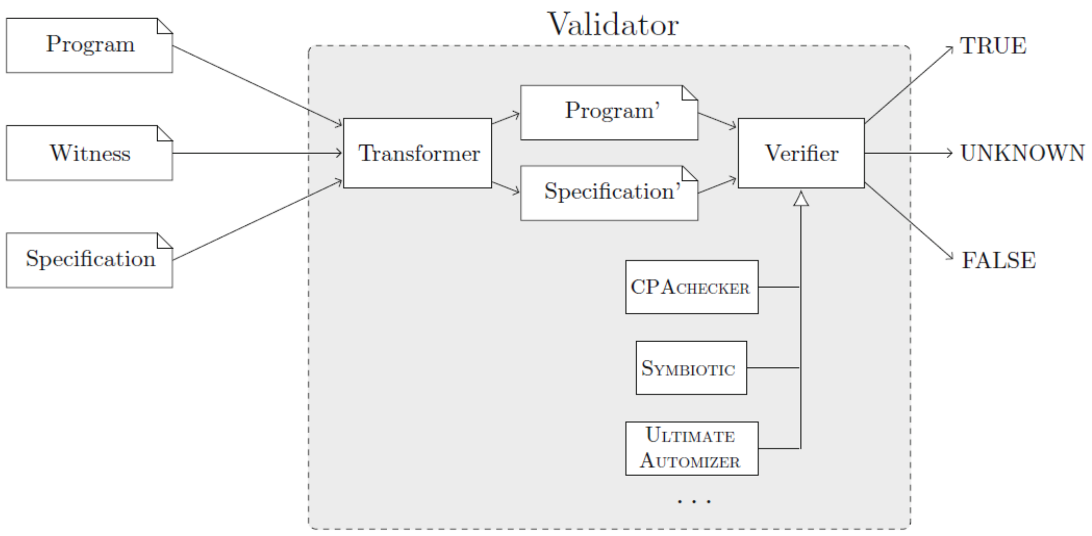


Figure 2.7: Architecture of the MetaVal

Source [23]

NitWit: This tool combines a C interpreter and a witness automaton to run validation. Figure 2.4.2.2 shows the validation technique.

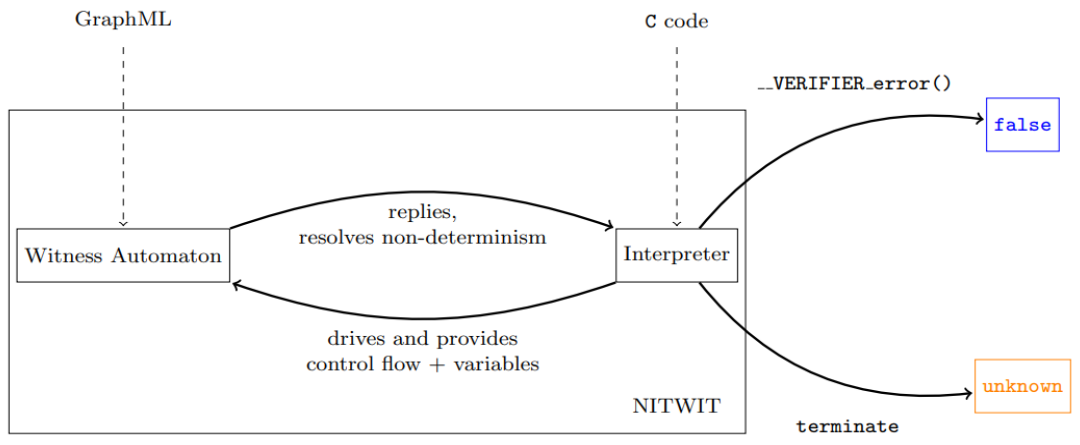


Figure 2.8: Architecture of the NitWit

Source [24]

As can be seen in Figure 2.8, the interpreter and witness automation (WA) work together. The C interpreter parses the program step by step, provides the current location of the program to WA, and then WA resolves the current non-deterministic variables. When the assumption from WA does not hold in the current statement, the validation ends, otherwise the validation result is unknown.

The C interpreter can execute each statement of the C program step by step without compiling the entire program, which is suitable for one-time execution.

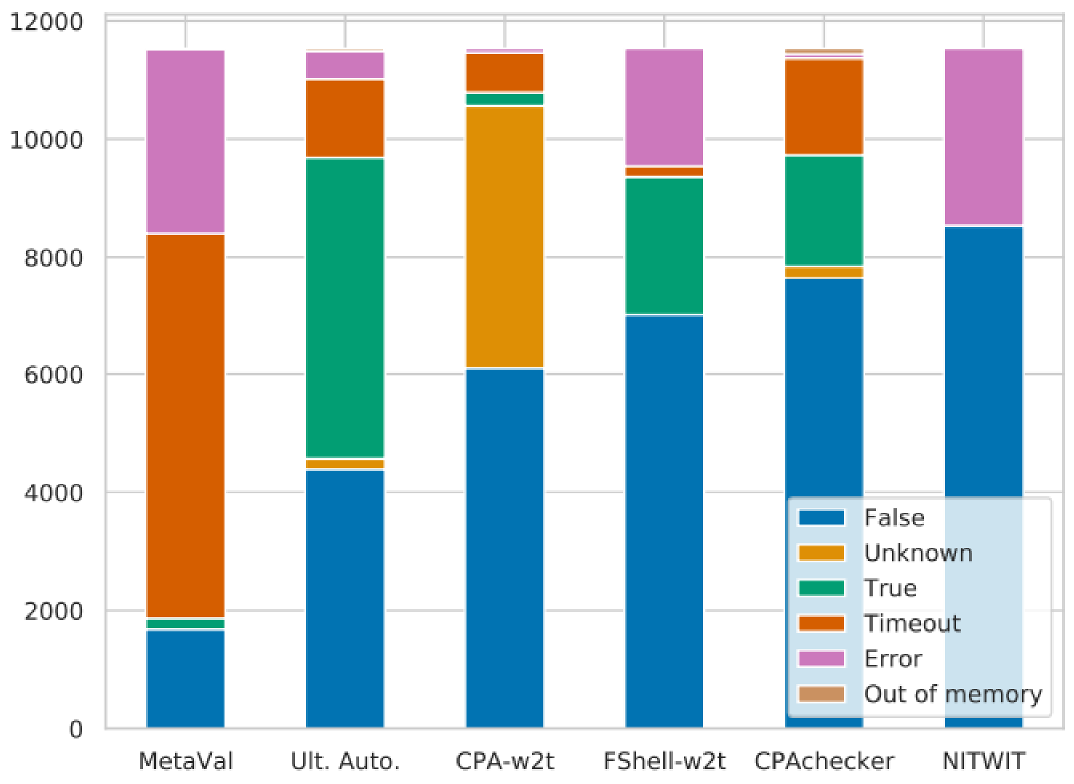


Figure 2.9: Validator outcomes on 11533 witnesses from SV-COMP 2020

Source [24]

These validators have different performances over the C benchmarks in SV-COMP. The Figure 2.9 shows the validation results of six C witness validators. In this figure, the blue bars indicate the number of successful violation-witness validations. Among the 11533 witnesses, MetaVal only has less than 2000 expected validation results, while the number of expected results of NitWit exceeds 8000, which is the highest number of the correct results among these tools.

This statistic shows that the validator based on the C interpreter is the most stable witness validator. Inspired by this technique, a similar interpreter for Java is Jshell, which is a command line program that available in JDK 9 and above. It can evaluate the entered Java statements or expressions and immediately output the result [25]. Although it seems to require many different technical and engineering work to introduce the Java interpreter into witness validation for Java, it is a very meaningful and challenging attempt.

2.4.3 Witness validation for Java

Since no stable witness validator for Java programs is contributed to the SV-COMP, it drives the need to develop a tool to validate the witnesses generated by Java verifiers.

The widely used validation method for the violation-witness is to check for counter-examples [5]. At present, Vi made an attempt to run the witness validation for Java programs, specifically validate the witnesses generated by JBMC [6]. Figure 2.10 shows the architecture of the extension, which contains the process of JBMC verification and witness validation.

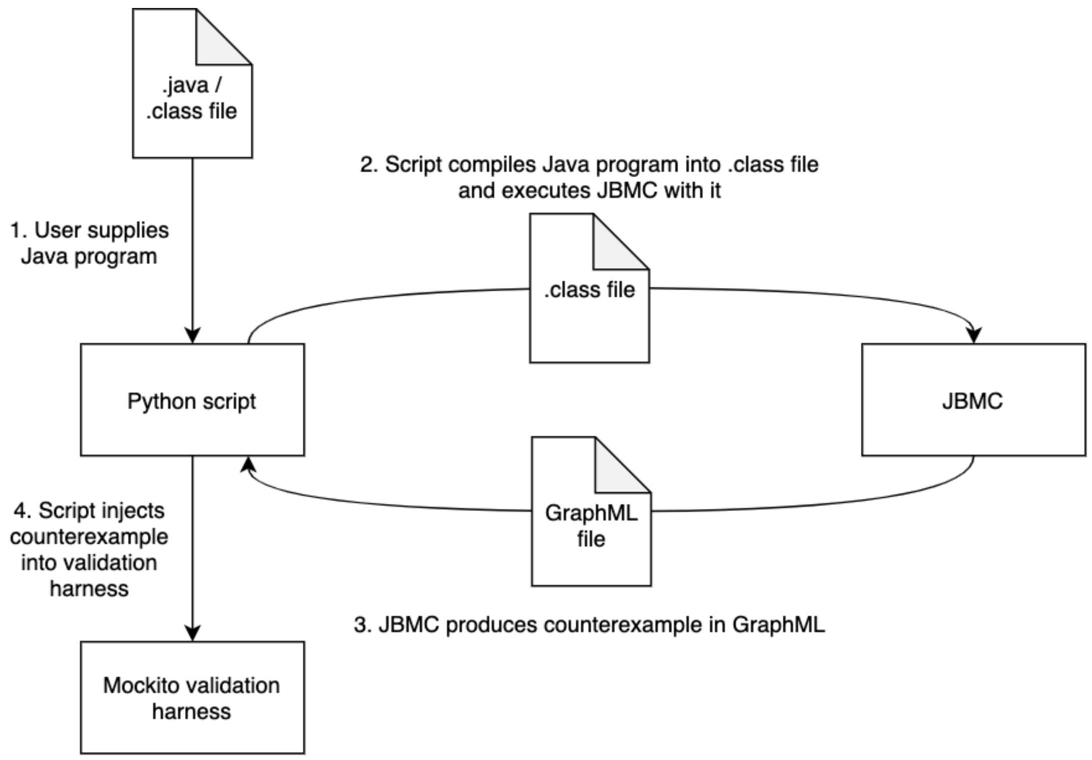
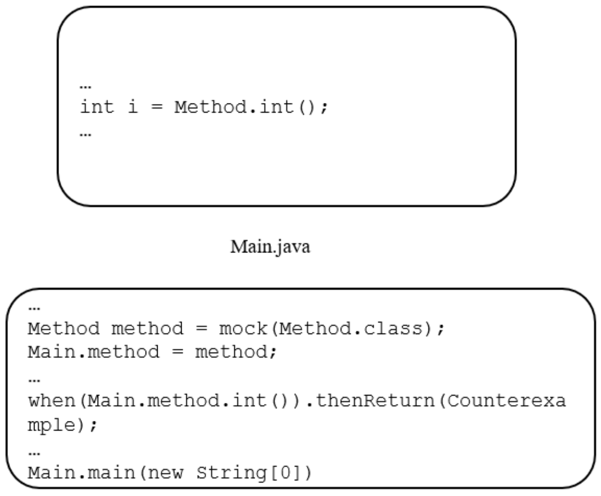

.java /
.class file
1. User supplies Java program
2. Script compiles Java program into .class file and executes JBMC with it
.class file
Python script
JBMC
GraphML file
3. JBMC produces counterexample in GraphML
4. Script injects counterexample into validation harness
Mockito validation harness

Figure 2.10: Architecture of the proposed extension by Vi

Source [6]

In this figure, the main program is a script written in Python. It takes a Java file or a compiled Java class file as input to run JBMC to detect vulnerabilities in the program, then extracts one counterexample from the witness file which is in the GraphML format, and inject it into a template to produce a Java unit test case. Specifically, in Java unit test, it introduces a unit testing framework Mockito [26] to simulate the class containing functions that can generate random values of various data types in the Java benchmarks, and uses the counterexample value as the return value of the function that generates the random value, thereby assigning the counterexample value to the source program variable.



```
…
int i = Method.int();
…
```

Main.java

```
…
Method method = mock(Method.class);
Main.method = method;
…
when(Main.method.int()).thenReturn(Counterexa
mple);
…
Main.main(new String[0])
```

Test with Mockito

Figure 2.11: Example of Witness validation via Mockito

Figure 2.11 illustrates an example of the validation algorithm using Mockito. In the `Main.java` program, `i` is a variable of int type, `Method.int()` is a function from a class that returns a random value of type int. A counterexample of `i` is given in the witness when a violation property was found by JBMC. In witness validation, the class `Method` is simulated by `mock()` function of Mockito, and when the `Meth-od.int()` function was called, it will return the value of the counterexample in the witness. Thus, the assignment of counterexample equals to the java statement `int i = counterexampleValue`. Finally, run the program of Java unit test with Mockito, If the same error attribute as that detected by JBMC is found, then it means that the counterexample is accepted and the validation is successful.

This extension provides a solution to do the witness validation for Java programs, but has a limited performance on the Java benchmarks in the SV-COMP.

Total benchmarks	Failed validation	String limits	Successful validation
177	109	40	28

Table 2.1: Result of witness validation of Vi's extension

As shown in Table 2.1, we summarized the validation results of this extension based on the data in his dissertation [6]. Among all 177 Java benchmarks, only 28 witnesses were successfully validated, 109 were not supported, and the remaining 40 were unable to be validated due to the restrictions of string manipulation by JBMC. The main reason why this extension does not support the witness validation of the 109 benchmarks is that the algorithm does not consider no or multiple calls to the functions in `Method` class.

In addition, this extension has many other limitations. First, it does not distinguish between validation of the correctness-witness and the violation-witness, but uses the same algorithm to perform it. However, the content structure of the two kinds of witnesses is different. In the correctness-witness, there is no data named assumption that represents the counterexample of the verifier hypothesis. Therefore, the validation of the correctness-witness is wrong, although it did not find an unexpected violation property. Because the validation process did not actually make use of the content of the edges in the correctness-witness. In fact, people mainly focus on the violation-witness validation to prove whether the verifier has generated a false alarm, and little attention is paid to the validation of the correctness-witness.

Second, it did not run all the 473 Java benchmarks in the SV-COMP. The Java benchmarks used are from the same branch of Java benchmarks in the SV-COMP, which are not sufficient to represent more complex Java programs in other branches of Java benchmarks in the SV-COMP. Thus, in order to have more sufficient data to analyse the performance of the extension, more testing should be performed against other Java benchmarks. Only when all Java benchmarks in the SV-COMP are tested can we analyse its validation ability based on the competition.

Third, As the program becomes more complex, its validation ability will be insufficient. There are two main flaws in this validation algorithm:

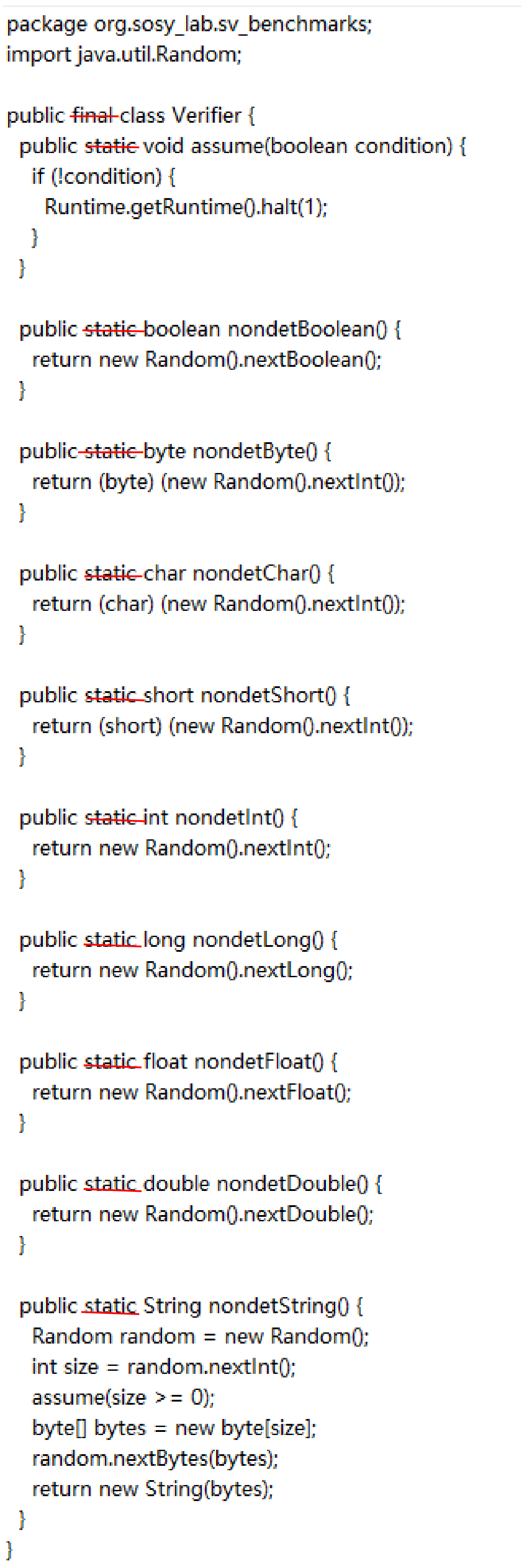
- **Counterexample extraction**

There are deficiencies in the way to find counterexamples in witnesses. A violation-witness will have counterexample values related to all variables in the program, but this algorithm only obtains one counterexample value, which will lead to incorrect instantiation of a Java program with multiple variables.

- **Counterexample assignment**

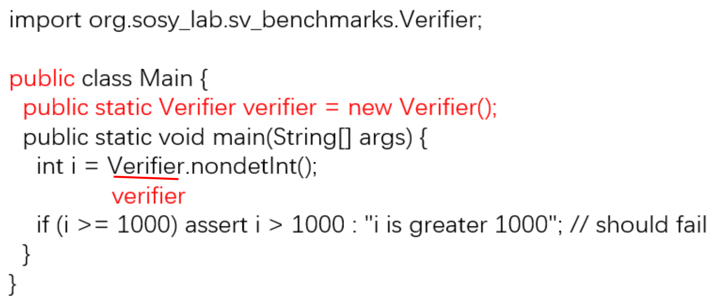
The method of assigning counterexample values is not rigorous and sometimes errors occur. It defaults that the counterexample value in witness is equal to the random value generated by the function in `Method` class, but this equation does not hold because sometimes the variable is not only determined by the `Method` class. For example, An `int` variable `i` in the Java code is equal to `Method.int()+1`, and the counterexample in witness is `i = 1`. Then if 1 is used as the return value of the function `Method.int()`, the value of `i` in the generated test case will be 2, and the counterexample assignment process will be wrong, which may cause the validator to not find the expected violation property.

Finally, this validation tool does not support multi-file input and multi-task execution. When a task involves multiple Java files, the algorithm needs to be optimized. In addition, due to the limitation that Mockito cannot simulate static methods, it is necessary to modify part of the code of `Method` class and Java benchmarks to run the tool (see Figure 2.12 and 2.13), which is very cumbersome for multitasking, and modifying the benchmarks is an illegal operation for the SV-COMP. Although the limitations of Mockito can be solved by using more advanced testing frameworks, such as the introduction of powermock to solve the problem that Mockito cannot simulate static methods [27], considering the flaws of the Java unit testing above, we will not introduce it in this experiment.



```
package org.sosy_lab.sv_benchmarks;
import java.util.Random;

public ~~final~~ class Verifier {
  public ~~static~~ void assume(boolean condition) {
    if (!condition) {
      Runtime.getRuntime().halt(1);
    }
  }

  public ~~static~~ boolean nondetBoolean() {
    return new Random().nextBoolean();
  }

  public ~~static~~ byte nondetByte() {
    return (byte) (new Random().nextInt());
  }

  public ~~static~~ char nondetChar() {
    return (char) (new Random().nextInt());
  }

  public ~~static~~ short nondetShort() {
    return (short) (new Random().nextInt());
  }

  public ~~static~~ int nondetInt() {
    return new Random().nextInt();
  }

  public ~~static~~ long nondetLong() {
    return new Random().nextLong();
  }

  public ~~static~~ float nondetFloat() {
    return new Random().nextFloat();
  }

  public ~~static~~ double nondetDouble() {
    return new Random().nextDouble();
  }

  public ~~static~~ String nondetString() {
    Random random = new Random();
    int size = random.nextInt();
    assume(size >= 0);
    byte[] bytes = new byte[size];
    random.nextBytes(bytes);
    return new String(bytes);
  }
}
```

Figure 2.12: Modified Verifier.java



```
import org.sosy_lab.sv_benchmarks.Verifier;

public class Main {
  public static Verifier verifier = new Verifier();
  public static void main(String[] args) {
    int i = Verifier.nondetInt();
            verifier
    if (i >= 1000) assert i > 1000 : "i is greater 1000"; // should fail
  }
}
```

Figure 2.13: Modified Main.java

In these two figures, the red parts indicate the modifications in the programs. In the SV-COMP, the `Method` class refers to `Verifier.java`, which contains static functions that generate random values. In these modifications, the `Verifier` class containing static functions is modified to a non-static ordinary class. When calling this class in the program, an object of this class is created first, and the class functions are called by this object, instead of calling the static functions of the original class.

Although the extension has some limits, it provides a method for Java witness validation by extracting counterexample values in witness and designing algorithms to assign counterexample values to new programs. In this project, we designed a new validator to better extract and utilize counterexample values in witness.

2.5 BenchExec

When there is no resource limit for the operation of the task, if an infinite loop occurs during operation, it may cause unlimited occupation of system resources and affects the subsequent tasks, which will also eventually lead to system crashes. Therefore, reasonable resource usage restrictions are necessary for batch execution of programs.

BenchExec is an open-source framework that can execute programs in batches and limit their resource usage. As the benchmark tool applied to the SV-COMP, BenchExec has successfully benchmarked thousands of benchmarks for dozens of participating tools in the competition. It can provide precise limits and measurements on running time, memory, CPU and other resources. In addition, it can also generate an interactive table from the benchmark result set, which displays the running status and resource usage of each task, and provides the function of viewing the running log of each task[7].

In this project, we introduced the BenchExec framework to manage the operation of the witness validator. At the same time, the validator can contribute to SV-COMP after being integrated into the BenchExec ecosystem.

2.6 Summary of background

Software testing plays a vital role in reducing program vulnerabilities and improving product quality in the software life cycle. Model checkers can be used as a supplement to traditional software testing methods to find program vulnerabilities, to test situations that are difficult to simulate artificially. But the key challenge in applying model checkers to practice is to check whether the witnesses that represent their verification results are valid. At present, the technology of witness validation is still immature, especially for the witness validation for Java programs, there is no stable algorithm yet.

3. Proposed Methodology

This chapter describes the methodology for implementing extensions to witness verification on top of JBMC. First, subchapter 3.1 introduces this extended system structure. Then, subchapter 3.2 introduces the implemented algorithm functions in detail by showing some code snippets. Finally, subchapter 3.3 introduces an example to show the results of the extension.

3.1 System Architecture

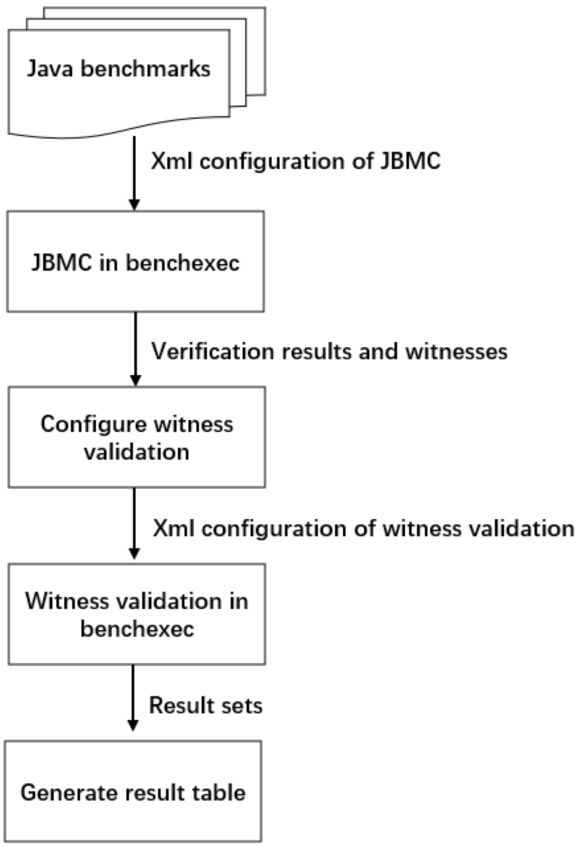

Java benchmarks
Xml configuration of JBMC
JBMC in benchexec
Verification results and witnesses
Configure witness validation
Xml configuration of witness validation
Witness validation in benchexec
Result sets
Generate result table

Figure 3.1: Architecture of the proposed extension

Figure 3.1 illustrates the overall structure and flow of the events. The user provides an xml configuration file for running JBMC in BenchExec as input via a Linux terminal, which defines the input Java files and resource usage limits.

The first step is to run JBMC in BenchExec. BenchExec will call JBMC to benchmark all the defined task files in the xml configuration file, output the results and generate the corresponding witness files. In particular, when a program has vulnerabilities and the output status of JBMC is verification failure, it will generate a violation-witness file, which represents the path to reach the assertion violation in the program. The code for this step is introduced in subchapter 3.2.1.1.

The second step is to configure the xml configuration file for witness validation running in BenchExec, which contains the JBMC task files set and the generated witnesses set. The code for this step is introduced in subchapter 3.2.1.2 and 3.2.1.3.

The third step is to run the witness validator in BenchExec. It traverses each task file and the corresponding witness, checks whether there is a violation-witness and validate it. Similarly, BenchExec will generate a result set of witness validation. The code for this step is introduced in subchapter 3.2.1.4.

The last step is to invoke the program provided by BenchExec to summarize the results of the two BenchExec runs into an html table, so that we can view the verification and validation results of each task and its corresponding resource usage. The code for this step is introduced in subchapter 3.2.1.5.

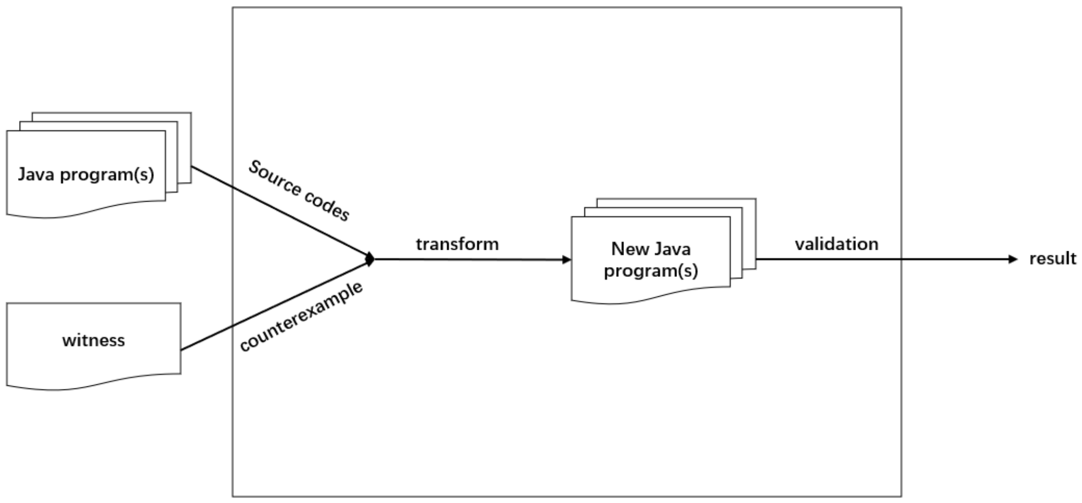


Figure 3.2: Architecture of the witness validator

Figure 3.2 illustrates the overall structure and flow of the witness validator. We need to take the Java program(s) and a witness as input. When the type of witness is violence-witness, it will provide counterexamples of the programs, and then inject the counterexamples into the corresponding position of the source programs to generate new programs. If the witness content is valid, then running the programs with counterexamples should reach the assertion violation statement, which shows that the verification result of the verifier is correct. The code for this process is introduced in subchapter 3.2.2.

3.2 Algorithms

3.2.1 Start script

This project first runs JBMC to verify the Java programs. Then, after all Java programs are verified, it will begin witness validation. The validation tool will read the witness file corresponding to each Java program. If it reads a violation-witness, then it will perform witness validation. Finally, after all witnesses are validated, it will display the results in a static html table, which shows both verification and validation results of these Java benchmarks. We automate this process through a program called startup script.

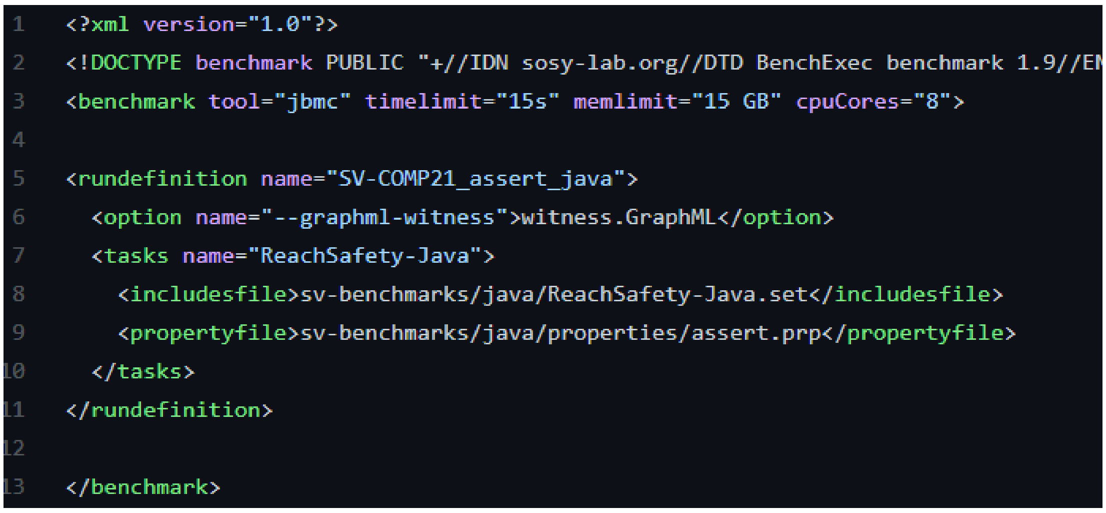
The startup script is a program written in python programming language, which can be run by executing the following commands in the ubuntu terminal:

```
sudo chmod +x execute.py
./execute.py ../File.xml
```

or

```
python3 execute.py ../File.xml
```

The first command is to set the script as an executable program in ubuntu. If you have set the startup script named `execute.py` as an executable program, you can execute the `./execute.py` command (the second command) to run it, otherwise, execute the latter `python3` command to run this script. For the last argument, the user needs to provide a file (see Figure 3.2.1) in the form of xml, which defines the execution commands of BenchExec, the resource limits and the task files to be executed [28].



```xml
<?xml version="1.0"?>
<!DOCTYPE benchmark PUBLIC "+//IDN sosy-lab.org//DTD BenchExec benchmark 1.9//E
<benchmark tool="jbmc" timelimit="15s" memlimit="15 GB" cpuCores="8">

<rundefinition name="SV-COMP21_assert_java">
  <option name="--graphml-witness">witness.GraphML</option>
  <tasks name="ReachSafety-Java">
    <includesfile>sv-benchmarks/java/ReachSafety-Java.set</includesfile>
    <propertyfile>sv-benchmarks/java/properties/assert.prp</propertyfile>
  </tasks>
</rundefinition>

</benchmark>
```

Figure 3.3: Configuration of JBMC in BenchExec

As can be seen from Figure 3.3, In the tag named BenchExec, we set the value of the attribute named tool to jbmc. The remaining three attributes are named timelimit, memlimit and cpuCores define resource limits. In this experiment, we limit the running time of each task to 15 seconds, the maximum memory usage to 15GB, and the number of cpu cores to 8. In the rundefinition area, we define the set of benchmarks to be executed in the tag named includesfile, and define the attribute value and witness file name in the tag named option.

The following subchapters show the details of the startup script.

3.2.1.1 Execution of the JBMC in BenchExec

```python
log = open('log.txt', 'a')
subprocess.Popen(['benchexec', sys.argv[1],'--no-compress-results'],stdout=log, stderr=log).wait()
```

```
3. with open("log.txt", "rt") as fin:
4.  for line in fin:
5.   print(line)
6. os.remove("log.txt")
7. tableXml = line[line.find(' ')+1:line.rfind('\n')]
```

Listing 3.1: Excerpt of script to benchmark JBMC in BenchExec

The main function of this code snippet is to execute the jbmc wrapper script in BenchExec. In this experiment, we do not run JBMC directly, but run the wrapper script, which is used as a tool for JBMC benchmarking in SV-COMP. Since JBMC can only support the verification of Java bytecode, it cannot read Java source code directly, so the programs in the Java benchmarks need to be compiled in advance. The script can parse the input property files to the actual JBMC, and return the running status codes for SV-COMP.

First, We execute the command through the `subprocess.Popen` module of python to execute the JBMC benchmark subprocess, and save and output the log content to the terminal. `sys.argv[1]` is the input xml definition file for running BenchExec, which defines the tool name jbmc, resource limits and task files to be executed. Then, in line 7, we save the string containing the file name and path of the result set to the variable `tableXml` in the last line of the log, which will be used to generate the result table later.

3.2.1.2 Extraction of witness filename and path

When the execution of the JBMC benchmark is finished, BenchExec will save all the generated witnesses to a result folder. We need to extract the folder path as input for the next run of the witness validator benchmark.

```
import xml.etree.ElementTree as ET
JBMC_taskFile_Root = ET.parse(sys.argv[1]).getroot()
witness_FileName = JBMC_taskFile_Root[0][0].text
witness_File = line[line.find('
')+1:line.rfind('results')] + 'files' + '/' + '${run-
definition_name}' + '/'+ '${taskdef_name}' + '/' +
witness_FileName
```

Listing 3.2: Excerpt of script to extract witness filename and path

In this code snippet, we first read the xml configuration file of jbmc running in BenchExec through `xml.etree.ElementTree` module of python to obtain the file name of the witnesses generated by jbmc. Then we define the complete file name and path of witnesses in line 4, which is used to provide input witnesses to the witness validator later.

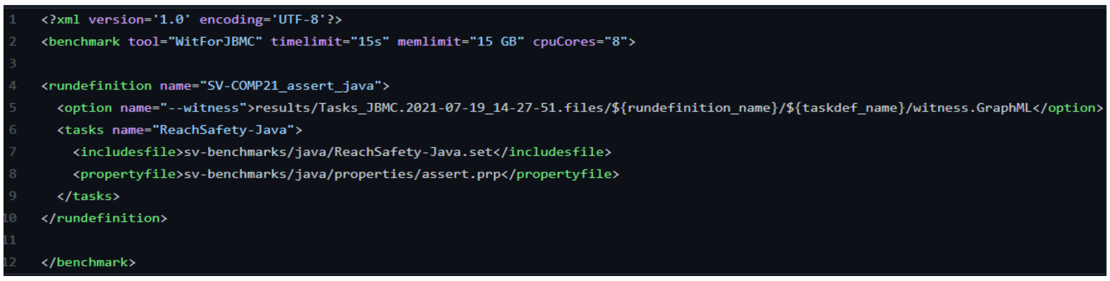
Each witness file generated will be saved to a specific directory separately in the result folder. This witness file set will be used for subsequent witness validation. In order to input the specific witness for each benchmark, we need to find the correspondence between each benchmark and the witness. We use the variables ${rundefinition_name} and ${taskdef_name} defined by BenchExec, which respectively define the current run definition and task-definition file that can be used to search for the corresponding witness file. As a result, we can find the witness for each Java benchmark according to the string variable `witness_File`.

3.2.1.3 XML configuration of the witness validator

```
1. task_Validation_Config =
   '../Tasks_JBMCWitnessValidator.xml'
2. copyfile(sys.argv[1], task_Validation_Config)
3. Validator_taskFile = ET.parse(task_Validation_Config)
4. Validator_taskFile_Root = Validator_taskFile.getroot()
5. Validator_taskFile_Root.set('tool','WitForJBMC')
6. option = Validator_taskFile_Root.find('rundefinition')
   .find('option')
7. option.set('name','--witness')
8. option.text = witness_File
9. Validator_taskFile.write(task_Validation_Config, en-
   coding='UTF-8', xml_declaration=True)
```

Listing 3.3: Excerpt of script to configure xml for witness validation

This code creates and configures the xml file that runs the witness validator in BenchExec. First, line 1 defines the file name and path of the xml configuration file for running the witness validator. Then, in line 2 to line 9, we make a copy of the xml configuration file for running jbmc in BenchExec and modify the content of the new xml configuration file to run the witness validator benchmark. We first change the tool name to `WitForJBMC`. `WitForJBMC` is the name of the witness validator we defined in BenchExec. then add the value of the string `witness_File` as an option in `rundefinition`, which contains the file name and path of the witness file. In this experiment, we use the same resource limits for both benchmark runs. These two tasks use the same Java benchmarks set, and there are two differences in the xml configuration: First, the input option for JBMC is used to generate a witness while the option for witness validator is to input the corresponding witness. Second, the tool names of the two configurations are different (see Figure 3.4).



```
<?xml version='1.0' encoding='UTF-8'?>
<benchmark tool="WitForJBMC" timelimit="15s" memlimit="15 GB" cpuCores="8">

<rundefinition name="SV-COMP21_assert_java">
  <option name="--witness">results/Tasks_JBMC.2021-07-19_14-27-51.files/${rundefinition_name}/${taskdef_name}/witness.GraphML</option>
  <tasks name="ReachSafety-Java">
    <includesfile>sv-benchmarks/java/ReachSafety-Java.set</includesfile>
    <propertyfile>sv-benchmarks/java/properties/assert.prp</propertyfile>
  </tasks>
</rundefinition>

</benchmark>
```

Figure 3.4: Configuration of witness validator in BenchExec

3.2.1.4 Execution of the witness validator in BenchExec

```
1. log = open('log.txt', 'a')
2. subprocess.Popen(['benchexec',
   '../Tasks_JBMCWitnessValidator.xml',    '--no-compress-
   results'],stdout=log).wait()
3. with open("log.txt", "rt") as fin1:
4.  for line1 in fin1:
5.   print(line1)
```

```
6. os.remove("log.txt")
7. tableXml1 = line1[line1.find(' ')+1:line1.rfind('\n')]
```

Listing 3.4: Excerpt of script to benchmark validator in BenchExec

The structure of this code snippet is the same as that of subchapter 3.2.1.1. In line 2, we execute the BenchExec command in Python subprocess and input the xml configuration file for the witness validator to benchmark. Similarly, we output the benchmark log and save the string containing the file name and path of the result set in the last line of the log to the variable `tableXml1`.

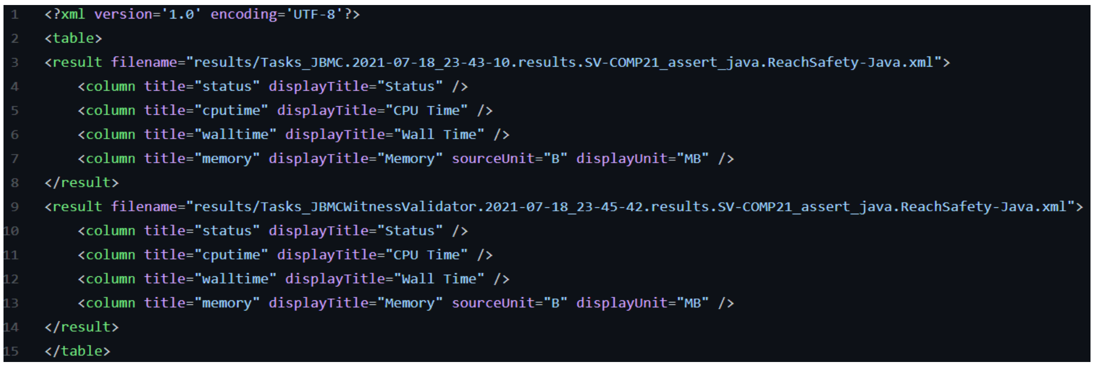
3.2.1.5 Generation of result table

After BenchExec runs, it will save the benchmark results to an xml file, and output the xml file name and path in the last line of the log. However, obtaining the log corresponding to each task result requires manually finding the corresponding folder, which is very cumbersome. Therefore, we need to visually display the results to view the running status and logs of each task. BenchExec provides a program called table-generator to read the xml result file to generate a beautiful and easy-to-view table. It is also applied to the SV-COMP to generate tables of all benchmark results. In this experiment, we also use the program to generate the result table.

```
tables = []
tables.append(tableXml)
tables.append(tableXml1)
table = ET.parse('table.xml')
table_Root = table.getroot()
for result,ta in zip(table_Root.iter('result'),tables):
 result.set('filename',ta)
table.write('table.xml',encoding='UTF-8',xml_declaration=True)
subprocess.Popen(['table-generator', '-x', 'table.xml']).wait()
```

Listing 3.5: Excerpt of script to generate the table

The function of this code snippet is to summarize the results of two BenchExec runs to generate a html and csv table. First, from line 1 to line 3, we store the two strings containing the xml file names and paths of the two running results into a list. Then, from line 4 to line 8, we write the elements of the list into the xml configuration file for the table-generator. Finally, in line 9, We execute the program table-generator with the input xml configuration file (see Figure 3.5) to generate the table.



```xml
<?xml version='1.0' encoding='UTF-8'?>
<table>
<result filename="results/Tasks_JBMC.2021-07-18_23-43-10.results.SV-COMP21_assert_java.ReachSafety-Java.xml">
    <column title="status" displayTitle="Status" />
    <column title="cputime" displayTitle="CPU Time" />
    <column title="walltime" displayTitle="Wall Time" />
    <column title="memory" displayTitle="Memory" sourceUnit="B" displayUnit="MB" />
</result>
<result filename="results/Tasks_JBMCWitnessValidator.2021-07-18_23-45-42.results.SV-COMP21_assert_java.ReachSafety-Java.xml">
    <column title="status" displayTitle="Status" />
    <column title="cputime" displayTitle="CPU Time" />
    <column title="walltime" displayTitle="Wall Time" />
    <column title="memory" displayTitle="Memory" sourceUnit="B" displayUnit="MB" />
</result>
</table>
```

Figure 3.5: Configuration of table-generator

As can be seen from this configuration, The table will display two result sets, which are the results of JBMC and witness validator. In addition, the results of each running task will show its running status, cpu time, wall time, and memory usage.

3.2.2 Witness Validator

The concept of this witness validator is to generate a new Java program based on the input source program and its corresponding witness file (which is similar to the MetaVal), and then run the new program to check whether it will reach the specified assertion error. It will assign the counterexample value of the variable in the witness to the corresponding variable in the new program. When all the non-deterministic variables in the program are assigned the specific counterexample value in the witness, there will be only one result after the program is run. Therefore, the key to designing the witness validation algorithm is to correctly extract and make use of counterexamples in witness.

Our approach is to do the violation-witness validation, since the validation algorithm extracts counterexamples from witnesses, which only exists in the violation-witnesses.

Witness validator for Java we designed is a program written in python programming language, which can be run by executing the following command in the terminal:
First:

```
sudo chmod +x Wit4JBMC.py
```

Second:

```
./Wit4JBMC.py --witness witness.GraphML *.java
```

or

```
./Wit4JBMC.py --witness witness.GraphML somepath
```

The first command is to set the witness validation script named `Wit4JBMC.py` as an executable program in ubuntu. It is necessary if the script needs to be run in BenchExec because BenchExec can only run executable tools while the script needs to be run using the python command. Then execute the `./ Wit4JBMC.py` command to run it. Otherwise, if you only need to run the witness validator without integrating it into BenchExec, execute the script `Wit4JBMC.py` by command `python3`:

```
python3 Wit4JBMC.py --witness witness.GraphML *.java
```

or

```
python3 Wit4JBMC.py --witness witness.GraphML somepath
```

When executing the commands to run the witness validator, you first need to enter the string `--witness` as an option name after the script name, then enter the relative path of the file containing the witness file name, and finally enter the relative path of all Java source files with file names involved in this task, or enter the relative directories of the parent folder of Java source files.

The following subchapters show the details of the validation script.

3.2.2.1 Extraction of Java source files

```
1. benchmarks_dir = []
2. for i in sys.argv[3:]:
3.  if '.java' in i:
4.   benchmarks_dir.append(i)
5.  else:
6.   for path, subdirs, files in os.walk(i):
7.    for name in files:
8.     if fnmatch(name, '*.java'):
9.    benchmarks_dir.append(os.path.join(path, name))
```

Listing 3.6: Excerpt of script to extract Java source files

The function of this code snippet is to save all input Java file names with their paths to a list. First of all, our algorithm considers two input formats for Java source files. On the one hand, if the input arguments are Java file names with their paths, then in lines 3 and 4, we store the matched argument in the form of `.java` to the list. On the other hand, if the input argument is the parent folder directory of the Java files, then from line 5 to line 9, we call the python function os.walk() to traverse the Java files in the folder and save the file names and paths into the list. Finally, in line 2, we traverse each argument starting from `sys.argv[3]` to ensure that all Java source files are saved. The purpose of saving them to a list is to facilitate subsequent traversal of each Java file.

3.2.2.2 Determination of witness file

```
from networkx import networkx as nx
witness_File_Dir = sys.argv[2]
witnessFile = nx.read_graphml(witness_File_Dir)
Witness = False
for violationKey in witnessFile.nodes(data = True):
 if 'isViolationNode' in violationKey[1]:
  Witness = True
```

Listing 3.7: Excerpt of script to determine witness

This code snippet checks if there is a violation node in the witness as a judgment condition for whether witness verification needs to be run. Sometimes when the JBMC verification result is true, there may be no content in the witness file. We call the `read_graphml()` function of the `networkx` package, which provides powerful manipulation on the graphs in xml format to read the graph format content of the witness file.

3.2.2.3 Counterexample extraction

When extracting counterexamples, the algorithm needs to consider the following data values in the edges in witness:

witness_type: The witness type. It is divided into correctness-witness and violation-witness. The algorithm determines whether the witness is a violation-witness, and if it is a correctness- witness, the witness validation is not performed [22].

originFileName: The name of the Java program given the counterexample. The algorithm judges whether the data value is the current program name, and if it is, the algorithm should extract the counterexample contained in this edge [22].

assumption.scope: The scope of the program corresponding to the counterexample. The algorithm judges whether the name of the program contained in the data value is the current program name, and if it is, the algorithm should extract the counterexample contained in this edge [22].

assumption: Counterexample to the scope of the current program. If the above conditions are all met, the data value is obtained as the value of the corresponding variable in the newly generated program [22].

startLine: The line number of the counterexample value in the program. If the above conditions are met, the data value is obtained as the line number of the corresponding variable in the newly generated program [22].

```
01.  def takeFirst(elem):
02.   return elem[0]
03.  if (Witness):
04.   witness_type = witnessFile.graph['witness-type']
05.   if (witness_type == 'violation_witness'):
06.    for benchmark in benchmarks_dir:
07.     new_benchmark_dir = ''
08.     counterexample = []
09.     for data in witnessFile.edges(data = True):
10.      if(data[2]['originFileName'] in benchmark and
         'assumption.scope' in data[2]):
11.       scope = data [2]['assumption.scope']
12.       startLine = data[2]['startline']
13.       if benchmark[benchmark.rfind('/')+1:benchmark
          .find('.java')] in scope or 'java' == scope:
14.        assumption = data [2]['assumption']
15.        counterexample.
           append(tuple((startLine,assumption)))
16.     counterexample.sort(key = takeFirst)
```

Listing 3.7: Excerpt of script to extract counterexamples

This code snippet extracts counterexample values of variables from witness. When the witness type is `violation_witness`, we extract counterexamples from the witness. The algorithm contains a double loop, the outer loop is to traverse each Java file name, and the inner loop is to traverse each edge of the witness, so as to traverse each edge of witness for each Java file in the list. In the inner loop, it finds the edge whose value of the attribute `originFileName` in the witness matches the Java file name. When there is a match, it extracts the values of the attributes `startLine` and `assumption`, and stores them in a list as a tuple.

After all counterexamples are extracted and stored, in line 16, we call the `sort()` function to sort the tuples in the list according to `startLine`, and arrange the tuples in the list in the ascending order of `startLine` natural number, which is convenient for instantiating the Java source code in line order. Figure 3 shows the sort principle.

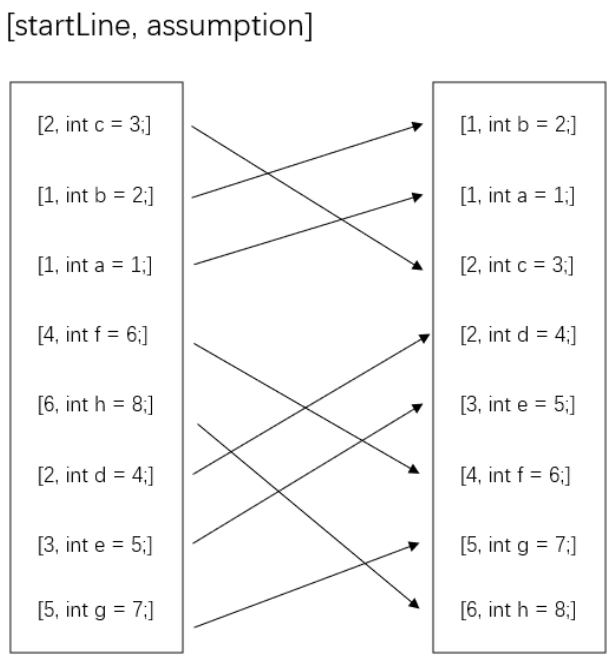


Figure 3.6: Rearrange the order of counterexamples

In the Figure, the tuples in the list on the left are listed in the order of their positions in the witness, and the tuples in the list on the right are arranged in ascending order according to the first element of each tuple, which is the row number.
The use of sorted tuples in the list is described in the next section.

3.2.2.4 Instantiation of source code

```
with open (benchmark,'rt') as fi:
 for line in fi:
  filename = benchmark[benchmark.rindex('/')+1:]
  if (line.strip().find('package') == 0):
   new_benchmark_dir = line.strip().replace('.','/').
   replace(';',' ').replace('package','').
   replace(' ','')
   if not os.path.exists(new_benchmark_dir):
    os.makedirs(new_benchmark_dir)
   Break
```

```
with open(benchmark,'r') as fii:
 with open(os.path.join(new_benchmark_dir,
  filename),'wt') as newfi:
  for position,line1 in enumerate(fii,1):
   line_new = line1
   if (len(counterexample) != 0):
    while (position == counterexample[0][0]):
     str = counterexample[0][1]
     if('++' in line1 or '--' in line1 or '*=' in line1):
      line_new = line1
     else:
      try:
        if(last_str(line1[:line1.index('=')].
        strip()) == str[:str.index('=')].strip()):
         if('&' not in str):
           line_new = line_new.replace
          (line_new[line_new.index('='):
          line_new.rindex(';')].strip(),str
          [str.index('='):str.rindex(';')].strip())
      except:
        line_new = line1
     counterexample.remove(counterexample[0])
     if (len(counterexample) == 0):
      Break
   if ' void ' in line1 and ' main' in line1 and
   'public  ' not in line1:
    line_new = 'public ' + line_new
   newfi.write(line_new)
```

Listing 3.8: Excerpt of script to assign counterexamples

In this code snippet, we create a new Java file by assigning counterexample values to variables corresponding to the Java source file. First of all, in order to ensure the normal operation of the compiled Java program, we standardize the relationship between the package definition in each Java source file and the relative directory of the actual file. From line 4 to line 8, we find the content after the keyword `package` in the Java code, call the function `os.makedirs()`, and use it as a relative directory for storing the newly instantiated Java file. Second, starting from line 9, we create a file in the new directory based on the name of the source Java file, and then copy the content of the source file line by line starting from the first line. At the same time, we traverse the list of counterexample tuples, when the current new program line number exists in the list of two-tuples, we assign values to the variables of the new Java code. From line 13 to line 26, if the variable name in this line of code is the same as the variable name of the counterexample corresponding to the line of the two-tuples list, then we assign the counterexample value to the variable in this line of code and delete the tuples in the list. Finally, for some Java benchmarks where `public` is not declared in the `main` function, so that it cannot be freely accessed by the JVM, from line 27 to line 29, we add the `public` keyword to the `main` function.

The algorithm skips two cases in the process of assigning values. The first one is variable increment and decrement, because they usually appear in Java loops, assignments may result in insufficient loop times and incorrect verification results. Because for a loop condition variable i, the algorithm will continuously iterate i to a new value until the value of i is the last iteration value, but the Java statements in the current condition is not executed during the iteration, and the generated program only contains the statement at the last loop iteration, and dose not completely traverse the loop of the source program. While the increment or decrement variables that do not appear in the loop do not need to be assigned, because its calculated value is the same as the counterexample value. The second is the assignment of `String` objects. Since JBMC has limits on `String` operations, it does not actually give valid counterexamples to nondeterministic String values.

3.2.2.5 Execution of instantiated code

```
01.  cmd = 'javac Main.java'
02.  subprocess.Popen(cmd,shell=True).wait()
03.  cmd1 = 'java -ea Main'
```

```
subprocess.Popen(cmd1,shell=True).wait()
```

Listing 3.9: Excerpt of script to execute the new program

This code snippet is to compile and run the new generated Java program. We call the `subprocess.Popen` module to run a subprocess that executes the Java command line. We execute the command `javac` to compile the Java program which contains the `main` function to generate bytecode, and then execute the command `java` to run the compiled codes. The -`ea` option must be added after the `java` command to output the assertion errors to reproduce the failure found by JBMC.

3.2.3 Tool Integration Module

In order to benchmark the witness validator and contribute it to the SV-COMP, it needs to be added to the tool module of BenchExec[29]. The following is the complete tool module information script written in Python.

```
import benchexec.tools.template
import benchexec.result as result
class Tool(benchexec.tools.template.BaseTool2):
 def executable(self, tool_locator):
  return tool_locator.find_executable("Wit4JBMC.py")
 def name(self):
  return "Wit4JBMC"
 def cmdline(self, executable, options, task, rlimits):
  return [executable] + options + list
    (task.input_files_or_identifier)
 def determine_result(self, run):
  output = run.output
  validation = 'unknown'
  for line in output:
   if 'Exception' in line:
    if 'AssertionError' in line:
     validation = 'false'
    else:
     validation = 'unknown'
    break
   else:
    validation = 'true'
  if validation == 'false':
   status = result.RESULT_FALSE_PROP
  elif validation == 'true':
   status = result.RESULT_TRUE_PROP
  else:
   status = result.RESULT_UNKNOWN
  return status
```

Listing 3.10: Excerpt of script to integrate the tool

As can be seen from this code that it mainly provides three functions:

Find the executable tool. the function `executable` finds the executable program named `Wit4JBMC.py`, which is the witness validation program.

Configure the command line to run the tool. The function `cmdline` will return the command to run witness verification. The command line includes the name of the executable program, the running option and input files. The command line examples of the validator are introduced in subchapter 3.2.2.

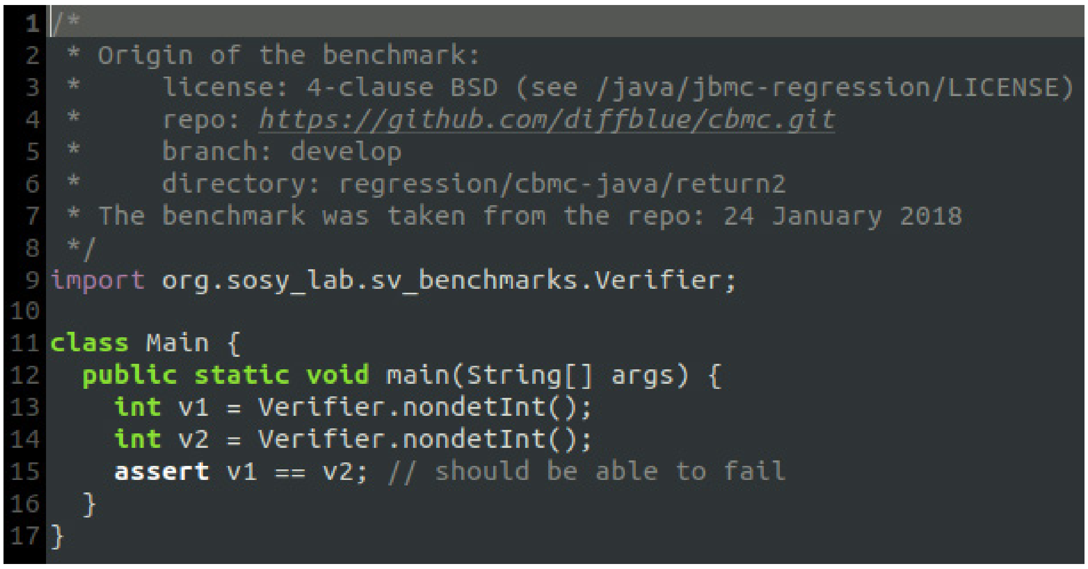
Confirm the result. The function `determine_result` will return the result of each task. We check every line in the result log of the witness validator. In the case of finding the `Exception` keyword, if there is also the `AssertionError` keyword, it means that the assertion error has been reproduced, otherwise it is another unknown error. When the `Exception` keyword does not appear, it means that the instantiated program ends normally and no exception is found, so the validation of the witness fails.

3.2.4 Complexity of the validation algorithm

Let *L* represent the number of programs in one task definition file, *M* represent the number of edges in the witness, and *N* represent the number of lines in each program. The validation algorithm contains a nested loop. The outer loop is to iterate each program, and the inner loop is to read each edge of the witness and write each line of the source program in the new program. Therefore, the complexity of the algorithm is represented as *O(L(M+N))*.

3.3 Illustrative Example

This sub-chapter outlines an illustrative example with a very simple Java program. This Java program is one of the Java benchmarks in SV-COMP, and the complete code screenshot is shown in Figure 3.7.



```
 1 /*
 2  * Origin of the benchmark:
 3  *     license: 4-clause BSD (see /java/jbmc-regression/LICENSE)
 4  *     repo: https://github.com/diffblue/cbmc.git
 5  *     branch: develop
 6  *     directory: regression/cbmc-java/return2
 7  * The benchmark was taken from the repo: 24 January 2018
 8  */
 9 import org.sosy_lab.sv_benchmarks.Verifier;
10
11 class Main {
12   public static void main(String[] args) {
13     int v1 = Verifier.nondetInt();
14     int v2 = Verifier.nondetInt();
15     assert v1 == v2; // should be able to fail
16   }
17 }
```

Figure 3.7: Screenshot of the Java program example

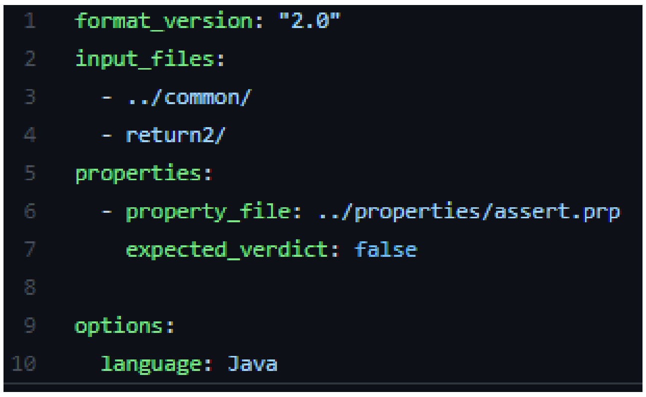
There is a vulnerability in this program. Since the values of variables `v1` and `v2` are random integer values generated by the `nondetInt()` method of the `verifier` class, they are non-deterministic. When the values of these two variables are not equal, the assertion statement in line 15 will fail. Therefore, when we run JBMC to verify this program, it should detect the vulnerability.

Next, we run this project, call the startup script to use JBMC to verify the program and do the witness verification.

In ubuntu terminal, input the command below:

```
./execute.py ../File.xml
```

Where `execute.py` is the name of the startup script and `File.xml` is the xml configuration file in which task file is the example program.

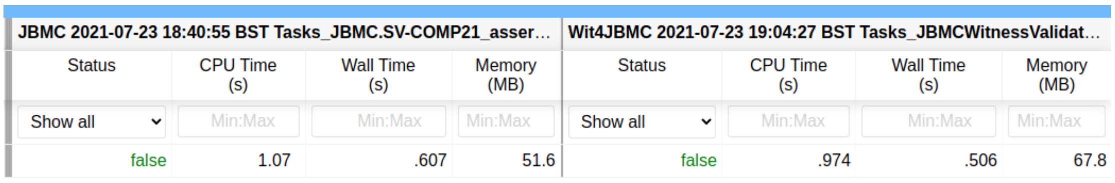


```
format_version: "2.0"
input_files:
  - ../common/
  - return2/
properties:
  - property_file: ../properties/assert.prp
    expected_verdict: false

options:
  language: Java
```

Figure 3.8: Screenshot of the task-definition file

The task file name defined in the `File.xml` is not the name of the Java program, but the name of the task-definition file (see Figure 3.8). The task-definition file is in the YAML language, which defines the configuration files and properties used. From the figure, tag `input_files` defines the parent directory path of the Java program to be input. In tag `properties`, `assert.prp` defined in subtag `property_file` indicates the specifications to be verified for the programs, and subtag `ex-pected_verdict` indicates the expected verification result.

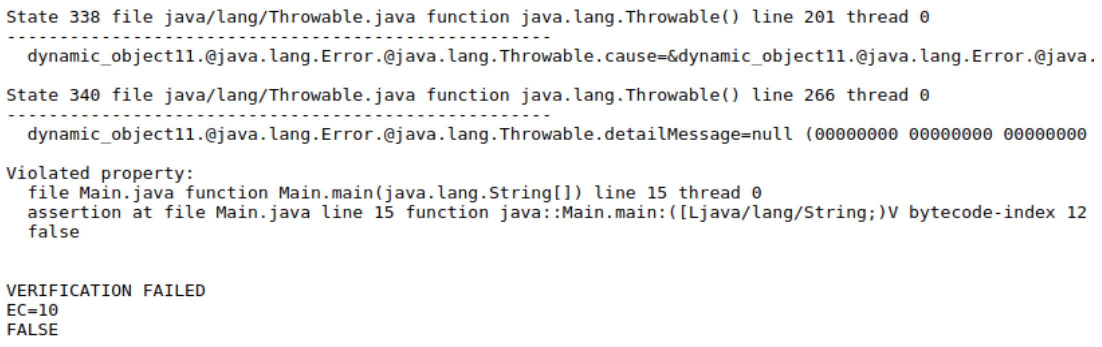
When the command is executed, we can see a newly generated html and csv file. The running result in the html file is shown in Figure 3.9.



JBMC 2021-07-23 18:40:55 BST Tasks_JBMC.SV-COMP21_asser...				Wit4JBMC 2021-07-23 19:04:27 BST Tasks_JBMCWitnessValidat...			
Status	CPU Time (s)	Wall Time (s)	Memory (MB)	Status	CPU Time (s)	Wall Time (s)	Memory (MB)
Show all	Min:Max	Min:Max	Min:Max	Show all	Min:Max	Min:Max	Min:Max
false	1.07	.607	51.6	false	.974	.506	67.8

Figure 3.9: Screenshot of the demo result table

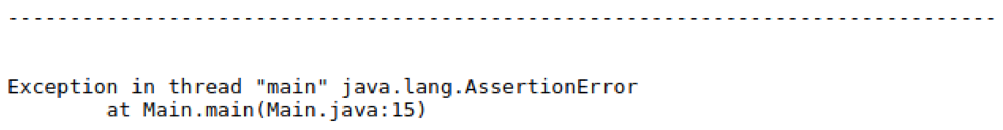
In this table, the first four columns are the running information of JBMC, and the last four columns are the running information of witness verification. These two pieces of information are separated by a thick line in the middle. We can read the running status of the task, cpu time, wall time and memory usage from the table. Among them, in the cell of the status information, the green font indicates that the running status is the expected result. At the same time, this cell provides a link to view the running log. We click on these two links respectively to view the JBMC log and the witness validation log.



```
State 338 file java/lang/Throwable.java function java.lang.Throwable() line 201 thread 0
----------------------------------------------------
  dynamic_object11.@java.lang.Error.@java.lang.Throwable.cause=&dynamic_object11.@java.lang.Error.@java.

State 340 file java/lang/Throwable.java function java.lang.Throwable() line 266 thread 0
----------------------------------------------------
  dynamic_object11.@java.lang.Error.@java.lang.Throwable.detailMessage=null (00000000 00000000 00000000

Violated property:
  file Main.java function Main.main(java.lang.String[]) line 15 thread 0
  assertion at file Main.java line 15 function java::Main.main:([Ljava/lang/String;)V bytecode-index 12
  false


VERIFICATION FAILED
EC=10
FALSE
```

Figure 3.10: Screenshot of the JBMC output

Figure 3.10 is the screenshot of JBMC running log. It can be seen that the verification failure and the position of the violation property in the program is printed in the log, which is the assertion statement in line 15.

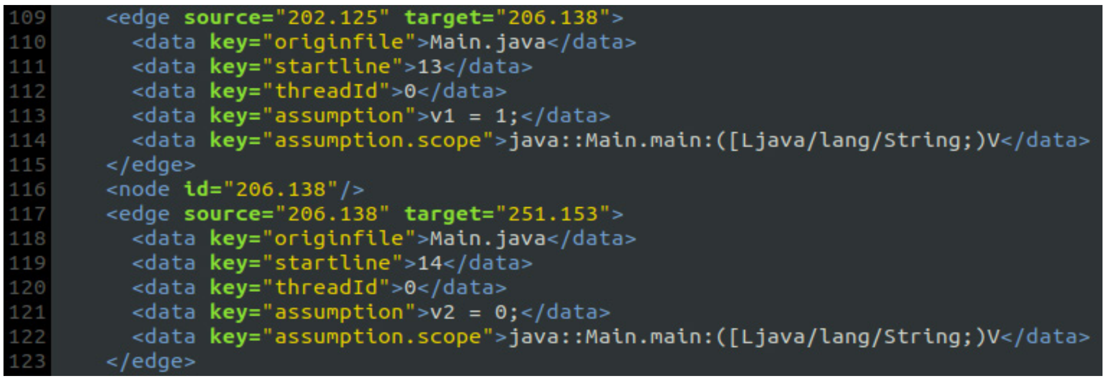


```
--------------------------------------------------------------------------------


Exception in thread "main" java.lang.AssertionError
        at Main.main(Main.java:15)
```

Figure 3.11: Screenshot of the witness validator output

Figure 3.11 is the screenshot of witness validator running log. It throws an exception because of the assertion error in line 15 of the program. It is consistent with the verifi-

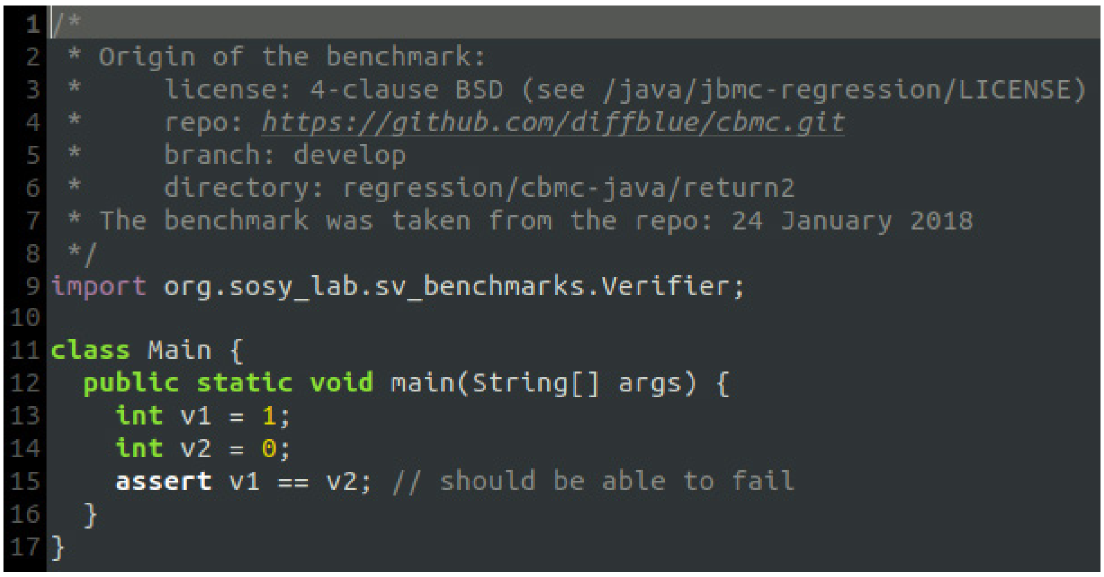
cation result of JBMC, which proves that the witness validator reached the specified program assertion error statement through the counterexamples in the witness.
In addition to the logs shown in the generated html table, we can also check the result folder generated by BenchExec to view other information, such as the witness and the newly generated programs.



```
    <edge source="202.125" target="206.138">
      <data key="originfile">Main.java</data>
      <data key="startline">13</data>
      <data key="threadId">0</data>
      <data key="assumption">v1 = 1;</data>
      <data key="assumption.scope">java::Main.main:([Ljava/lang/String;)V</data>
    </edge>
    <node id="206.138"/>
    <edge source="206.138" target="251.153">
      <data key="originfile">Main.java</data>
      <data key="startline">14</data>
      <data key="threadId">0</data>
      <data key="assumption">v2 = 0;</data>
      <data key="assumption.scope">java::Main.main:([Ljava/lang/String;)V</data>
    </edge>
```

Figure 3.12: Screenshot of the witness content

Figure 3.12 is the screenshot of part of the content of the witness file. The content corresponding to the key `originfile` is the Java program name, the content corresponding to the key `startline` is the line number of the statement in the program, and the content corresponding to the key `assumption` is the hypothetical variable value to reach the assertion violation, which is called a counterexample. In this example, the counterexamples are `v1 = 1` and `v2 = 0`.



```
/*
 * Origin of the benchmark:
 *     license: 4-clause BSD (see /java/jbmc-regression/LICENSE)
 *     repo: https://github.com/diffblue/cbmc.git
 *     branch: develop
 *     directory: regression/cbmc-java/return2
 * The benchmark was taken from the repo: 24 January 2018
 */
import org.sosy_lab.sv_benchmarks.Verifier;

class Main {
  public static void main(String[] args) {
    int v1 = 1;
    int v2 = 0;
    assert v1 == v2; // should be able to fail
  }
}
```

Figure 3.13: Screenshot of the new Java program

Figure 3.13 is the screenshot of part of the new Java program. As can be seen from lines 13 and 14, the counterexamples in witness have been successfully assigned to the new program. We can manually check that the counterexamples were successfully injected to this new program and the program will throw an exception of assertion error in 15 lines.

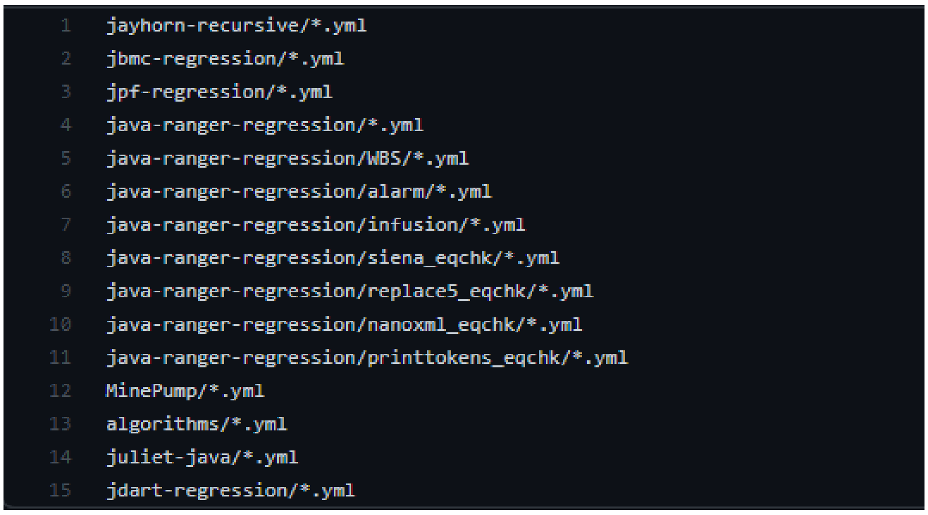
The above example introduces how this project works by entering a very simple Java program. The next chapter will introduce the test results of all Java benchmarks in the SV-COMP, some of which are much more complicated than this example.

4. Experimental Evaluation

This chapter introduces the experimental results of the project. First, subchapter 4.1 introduces the Java benchmarks used in this project. Then, subchapter 4.2 introduces the software and hardware environment information of the project, project directory, and suggestions before starting the experiment. Next, subchapter 4.3 introduces the objectives of the experimental evaluation. After that, subchapter 4.4 and 4.5 introduces the results in detail, as well as a detailed analysis of the results. Finally, subchapter 4.6 summaries the experimental conclusions.

4.1 Benchmarks

In this experiment, the benchmarks used are all Java benchmarks in the SV-COMP. For each Java benchmark, Java programs need to be listed in their corresponding task definition files.



```
jayhorn-recursive/*.yml
jbmc-regression/*.yml
jpf-regression/*.yml
java-ranger-regression/*.yml
java-ranger-regression/WBS/*.yml
java-ranger-regression/alarm/*.yml
java-ranger-regression/infusion/*.yml
java-ranger-regression/siena_eqchk/*.yml
java-ranger-regression/replace5_eqchk/*.yml
java-ranger-regression/nanoxml_eqchk/*.yml
java-ranger-regression/printtokens_eqchk/*.yml
MinePump/*.yml
algorithms/*.yml
juliet-java/*.yml
jdart-regression/*.yml
```

Figure 4.1: Task set screenshot

Figure 4.1 shows the task set used to run the project, which contains all the task definition files in the **.yml* format in these directories.

4.2 Setup

4.2.1 Environment setup

4.2.1.1 Java environment installation

Java is available for download at:
https://www.java.com/en/download/manual.jsp

4.2.1.2 Python environment installation

Python is available for download at:
https://www.python.org/downloads/

In our experiment, to ensure the correct work of BenchExec, Python needs to be version 3.6 or higher. Also make sure that the `networkx` package and `pip3` package are installed.

4.2.1.3 JBMC and the wrapper script

JBMC is available at the Github repository:
https://github.com/diffblue/cbmc/tree/develop/jbmc

JBMC wrapper script is available at the Github repository:
https://github.com/diffblue/cprover-sv-comp

It is recommended to install CBMC first, and JBMC should be installed before installing the wrapper script.

In addition, a complied and ready-to-use archive is available at the SV-COMP page:

https://gitlab.com/sosy-lab/sv-comp/archives-2021/raw/svcomp21/2021/jbmc.zip

4.2.1.4 BenchExec installation

BenchExec is available at the Github repository:
https://github.com/sosy-lab/benchexec

BenchExec provides a variety of installation packages. But it must be noted that since we will add a tool information module of the witness validator to BenchExec, you need to download the development version of BenchExec (source code) instead of the installation packages.

4.2.1.5 Proposed extension

The implementation in this project is available at the Github repository:
https://github.com/Anthonysdu/MSc-project

4.2.1.6 Java benchmarks

The complete Java benchmarks used in this project are available at the Github repository:
https://github.com/sosy-lab/sv-benchmarks/tree/master/java

4.2.2 Environment versions

The software and hardware used in this project are as follows:

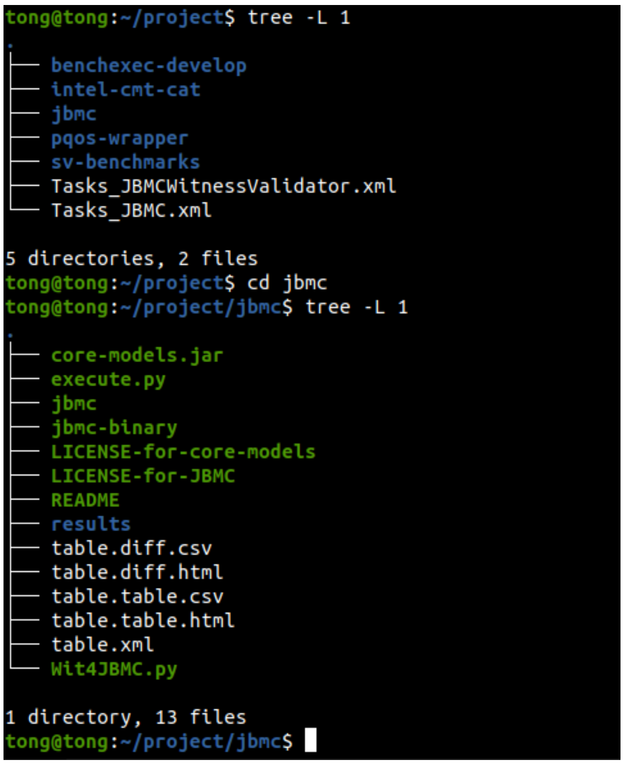
- JDK/JRE (version 1.8.0_292)
- python3 (version 3.8.10)
- JBMC (version 5.17.0)
- pip3 (version 20.0.2)
- BenchExec (version 3.8-dev)
- table-generator (version 3.8-dev)

- PQoS (version 4.1.0)
- Model: LENOVO_MT_82AV_BU_idea_FM_Legion Y7000 2020
- Operating System: Ubuntu 20.04.2 LTS.
- CPU: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz,12 cores
- RAM: 16 GB

The Linux operating system is required because BenchExec cannot work on other operating systems [7].

4.2.3 Running the tests

The directory tree of this project is illustrated in Figure 4.2.



```
tong@tong:~/project$ tree -L 1
.
├── benchexec-develop
├── intel-cmt-cat
├── jbmc
├── pqos-wrapper
├── sv-benchmarks
├── Tasks_JBMCWitnessValidator.xml
└── Tasks_JBMC.xml

5 directories, 2 files
tong@tong:~/project$ cd jbmc
tong@tong:~/project/jbmc$ tree -L 1
.
├── core-models.jar
├── execute.py
├── jbmc
├── jbmc-binary
├── LICENSE-for-core-models
├── LICENSE-for-JBMC
├── README
├── results
├── table.diff.csv
├── table.diff.html
├── table.table.csv
├── table.table.html
├── table.xml
└── Wit4JBMC.py

1 directory, 13 files
tong@tong:~/project/jbmc$
```

Figure 4.2: Project directory tree

Here is a basic introduction to the subdirectories under the project directory:

Benchexec-develop: The directory where the BenchExec installation version package is stored.

intel-cmt-cat: The directory where additional library (Intel PQoS) for BenchExec is stored.

jbmc: The directory where the JBMC wrapper script is stored.

pqos-wrapper: The directory where additional interface for BenchExec to use Intel PQoS is stored.

sv-benchmarks: The directory where the Java benchmarks are stored.

Tasks_JBMCWitnessValidator.xml: Xml configuration file to run witness validator in BenchExec.

Tasks_JBMC.xml: Xml configuration file to run JBMC in BenchExec.

In the jbmc folder, in addition to the original files, the newly created files are as follows:

execute.py: The startup script of the project.

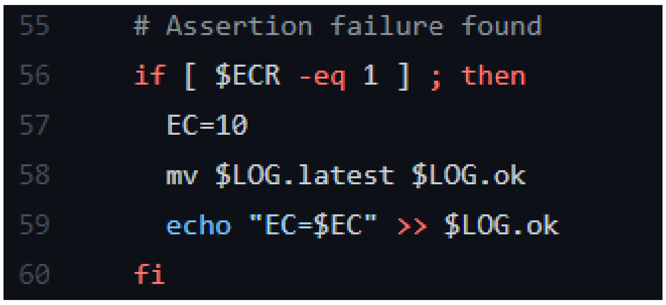
results: The directory where all the results by BenchExec are saved.

wit4JBMC.py: The script of the witness validator.

table.xml: Xml configuration to generate the result tables.

table.*.csv/html: Generated html and csv tables.

In addition, the new tool-info module we introduced in the previous chapter named `WitForJBMC.py` should be saved into the relevant BenchExec tools directory [29].



```
# Assertion failure found
if [ $ECR -eq 1 ] ; then
  EC=10
  mv $LOG.latest $LOG.ok
  echo "EC=$EC" >> $LOG.ok
fi
```

Figure 4.3: Code segment of JBMC Wrapper Script

Before running the experiment, a modification to the JBMC wrapper script needs to be made. The wrapper script parses input Java programs for JBMC. However, it uses the JVM to check the assertion properties of the program before sending the compiled bytecode to JBMC. If an assertion error is found, the program error is directly returned without calling JBMC to check the program, so there will be no witness file generated by JBMC.

As shown in Figure 4.3, it is part of the code of the wrapper script. In order to generate more violation-witness files for witness validation, we should comment the code in the figure so as to ensure that all programs will be passed to JBMC.

4.3 Objectives

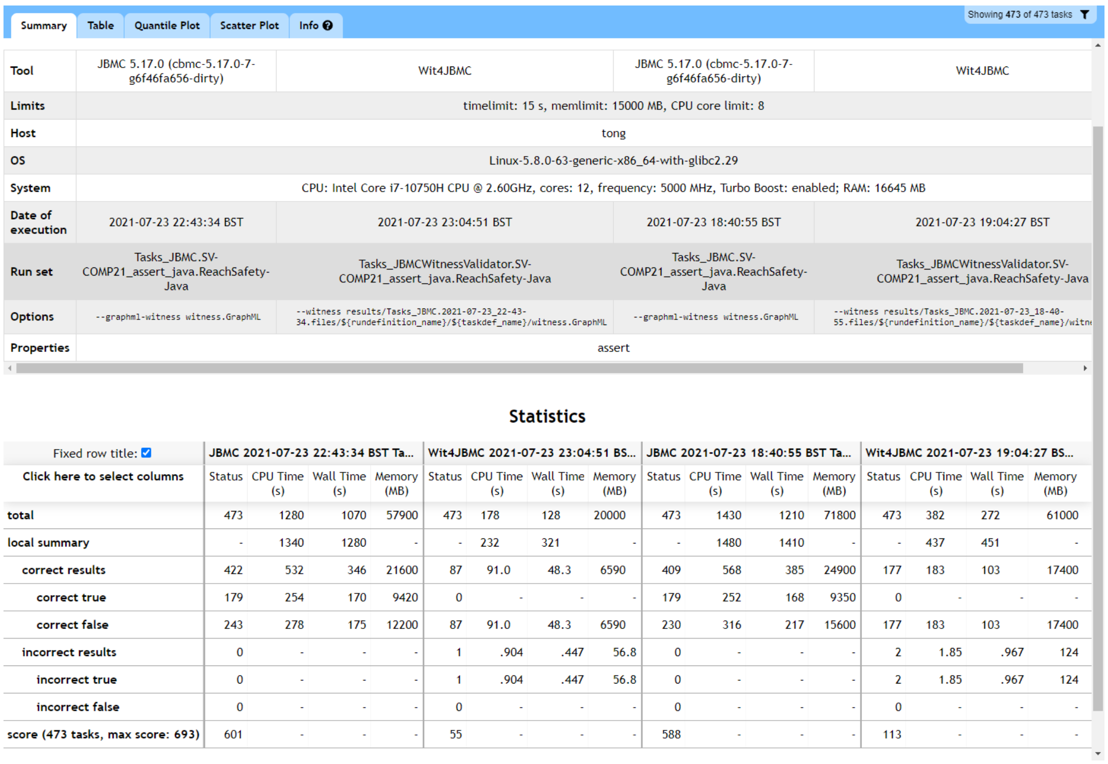
The specific objectives of this experimental evaluation are:

- Obtain the benchmark results of JBMC and the witness validator.
- Analyse the verification results of JBMC.
- Analyse the validation results of the witness validator.

4.4 Results

4.4.1 Result statistics

In order to compare the effect of the code of the JBMC wrapper script on the experiment after being commented, we conducted two experiments. One experiment kept the packaging script unchanged, and the other experiment commented out the code.



Tool	JBMC 5.17.0 (cbmc-5.17.0-7-g6f46fa656-dirty)	Wit4JBMC	JBMC 5.17.0 (cbmc-5.17.0-7-g6f46fa656-dirty)	Wit4JBMC
Limits	timelimit: 15 s, memlimit: 15000 MB, CPU core limit: 8			
Host	tong			
OS	Linux-5.8.0-63-generic-x86_64-with-glibc2.29			
System	CPU: Intel Core i7-10750H CPU @ 2.60GHz, cores: 12, frequency: 5000 MHz, Turbo Boost: enabled; RAM: 16645 MB			
Date of execution	2021-07-23 22:43:34 BST	2021-07-23 23:04:51 BST	2021-07-23 18:40:55 BST	2021-07-23 19:04:27 BST
Run set	Tasks_JBMC.SV-COMP21_assert_java.ReachSafety-Java	Tasks_JBMCWitnessValidator.SV-COMP21_assert_java.ReachSafety-Java	Tasks_JBMC.SV-COMP21_assert_java.ReachSafety-Java	Tasks_JBMCWitnessValidator.SV-COMP21_assert_java.ReachSafety-Java
Options	`--graphml-witness witness.GraphML`	`--witness results/Tasks_JBMC.2021-07-23_22-43-34.files/${rundefinition_name}/${taskdef_name}/witness.GraphML`	`--graphml-witness witness.GraphML`	`--witness results/Tasks_JBMC.2021-07-23_18-40-55.files/${rundefinition_name}/${taskdef_name}/witn`
Properties	assert			

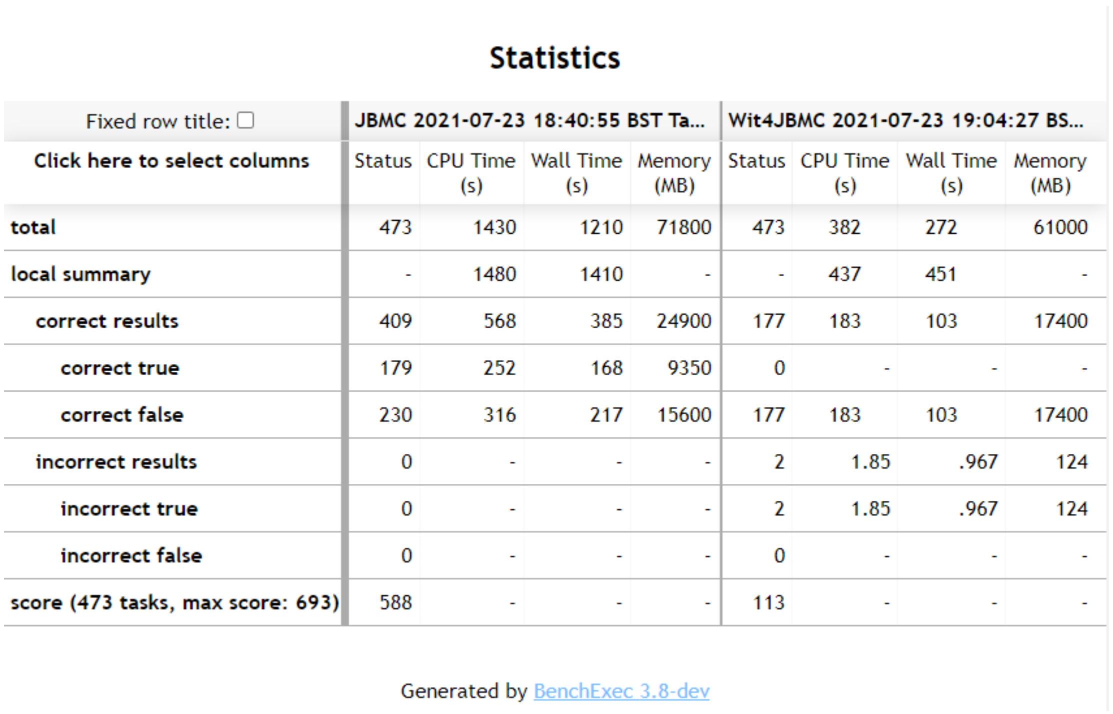
Statistics

Fixed row title: ✓	JBMC 2021-07-23 22:43:34 BST Ta...				Wit4JBMC 2021-07-23 23:04:51 BS...				JBMC 2021-07-23 18:40:55 BST Ta...				Wit4JBMC 2021-07-23 19:04:27 BS...			
Click here to select columns	Status	CPU Time (s)	Wall Time (s)	Memory (MB)	Status	CPU Time (s)	Wall Time (s)	Memory (MB)	Status	CPU Time (s)	Wall Time (s)	Memory (MB)	Status	CPU Time (s)	Wall Time (s)	Memory (MB)
total	473	1280	1070	57900	473	178	128	20000	473	1430	1210	71800	473	382	272	61000
local summary	-	1340	1280	-	-	232	321	-	-	1480	1410	-	-	437	451	-
correct results	422	532	346	21600	87	91.0	48.3	6590	409	568	385	24900	177	183	103	17400
correct true	179	254	170	9420	0	-	-	-	179	252	168	9350	0	-	-	-
correct false	243	278	175	12200	87	91.0	48.3	6590	230	316	217	15600	177	183	103	17400
incorrect results	0	-	-	-	1	.904	.447	56.8	0	-	-	-	2	1.85	.967	124
incorrect true	0	-	-	-	1	.904	.447	56.8	0	-	-	-	2	1.85	.967	124
incorrect false	0	-	-	-	0	-	-	-	0	-	-	-	0	-	-	-
score (473 tasks, max score: 693)	601	-	-	-	55	-	-	-	588	-	-	-	113	-	-	-

Figure 4.4: Statistics of two experimental results

Figure 4.4 provides a summary of the results of the two experiments. The top inside this figure shows the benchmark setup, which contains information such as tool name, resource limit, host name, operating system, hardware information, date of execution, run task set, etc. The bottom inside the figure shows the statistics of the running results. The statistical results have four columns: the first column is the benchmark results of JBMC, and its wrapper script remains unchanged. The second column is the validation results of the witnesses generated by JBMC. The third column is also the benchmark results of JBMC, but its wrapper script was modified. The fourth column is the validation results of the witnesses generated by subsequent JBMC.

Although it can be seen from the first and third columns that the JBMC benchmark using the modified wrapper script reduced 13 correct results and its score also dropped from 601 to 588, as can be seen from the second and fourth columns, the correct results of the latter are approximately twice that of the former, which means that running the modified wrapper script for benchmarking produces more witness files.



Statistics

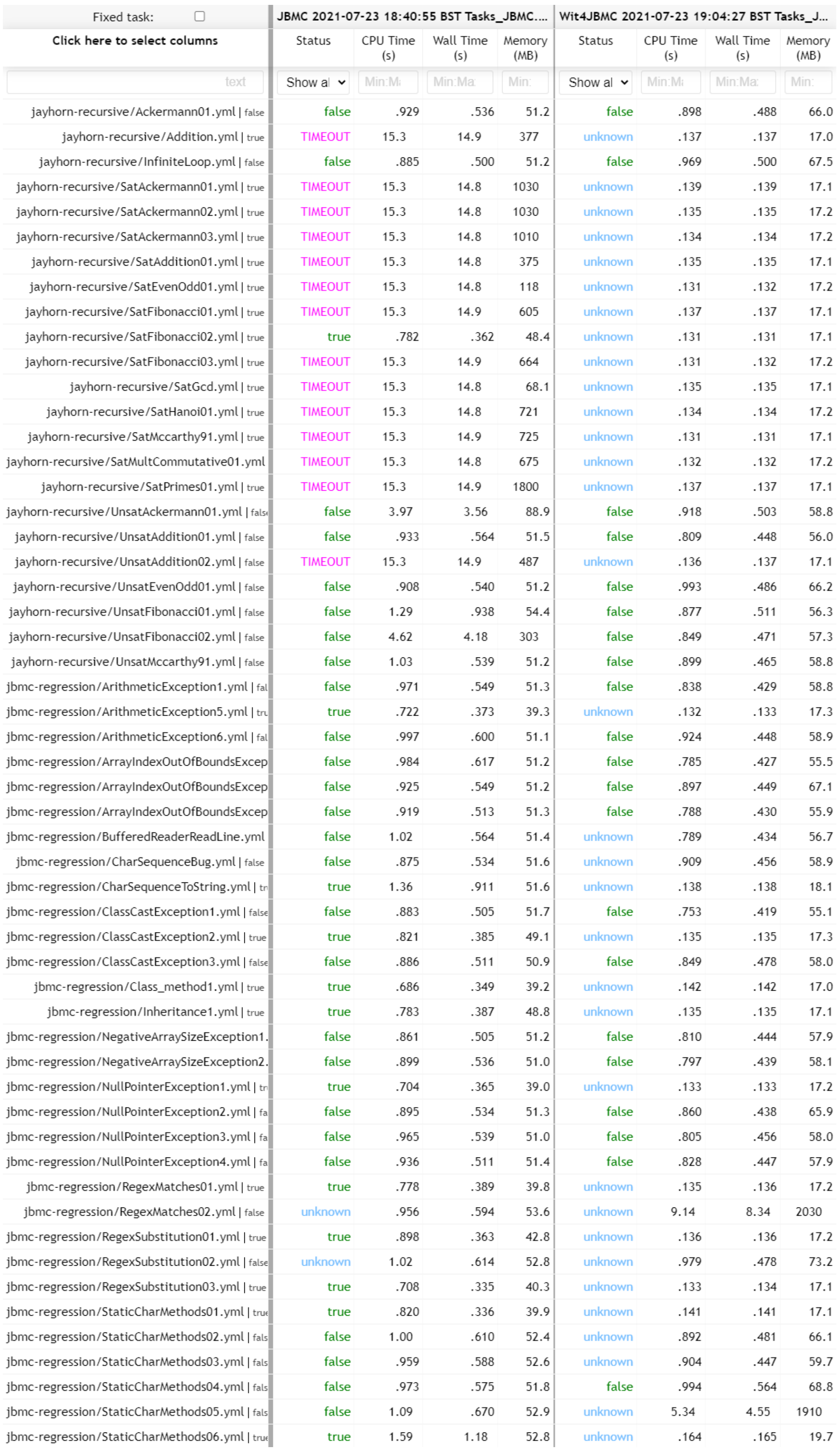
Fixed row title: ☐	JBMC 2021-07-23 18:40:55 BST Ta...				Wit4JBMC 2021-07-23 19:04:27 BS...			
Click here to select columns	Status	CPU Time (s)	Wall Time (s)	Memory (MB)	Status	CPU Time (s)	Wall Time (s)	Memory (MB)
total	473	1430	1210	71800	473	382	272	61000
local summary	-	1480	1410	-	-	437	451	-
correct results	409	568	385	24900	177	183	103	17400
correct true	179	252	168	9350	0	-	-	-
correct false	230	316	217	15600	177	183	103	17400
incorrect results	0	-	-	-	2	1.85	.967	124
incorrect true	0	-	-	-	2	1.85	.967	124
incorrect false	0	-	-	-	0	-	-	-
score (473 tasks, max score: 693)	588	-	-	-	113	-	-	-

Generated by BenchExec 3.8-dev

Figure 4.5: Statistics of single experimental results

Figure 4.5 shows the JBMC benchmark results and witness validator benchmark results after modifying the JBMC wrapper script. The left side of the table in the figure is the statistical results of JBMC. For all of these 473 Java benchmarks, JBMC correctly verified 409 of them, and the remaining 64 verification results are unknown. Among the 409 correct results, 179 corresponding programs have no violation attributes and the verification results are true, while the other 230 are found to have the violation attributes of the programs, and the verification results are false.

The right side of the table in the figure is the statistical results of the witness validation run. Regarding the 230 benchmarks with violation attributes verified by JBMC against these 473 Java benchmarks, the witness validator successfully validated 177 with 2 incorrect and 51 unknown results.



Fixed task: ☐	JBMC 2021-07-23 18:40:55 BST Tasks_JBMC....				Wit4JBMC 2021-07-23 19:04:27 BST Tasks_J...			
Click here to select columns	Status	CPU Time (s)	Wall Time (s)	Memory (MB)	Status	CPU Time (s)	Wall Time (s)	Memory (MB)
text	Show al	Min:Ma	Min:Ma	Min:	Show al	Min:Ma	Min:Ma	Min:
jayhorn-recursive/Ackermann01.yml | false	false	.929	.536	51.2	false	.898	.488	66.0
jayhorn-recursive/Addition.yml | true	TIMEOUT	15.3	14.9	377	unknown	.137	.137	17.0
jayhorn-recursive/InfiniteLoop.yml | false	false	.885	.500	51.2	false	.969	.500	67.5
jayhorn-recursive/SatAckermann01.yml | true	TIMEOUT	15.3	14.8	1030	unknown	.139	.139	17.1
jayhorn-recursive/SatAckermann02.yml | true	TIMEOUT	15.3	14.8	1030	unknown	.135	.135	17.2
jayhorn-recursive/SatAckermann03.yml | true	TIMEOUT	15.3	14.8	1010	unknown	.134	.134	17.2
jayhorn-recursive/SatAddition01.yml | true	TIMEOUT	15.3	14.8	375	unknown	.135	.135	17.1
jayhorn-recursive/SatEvenOdd01.yml | true	TIMEOUT	15.3	14.8	118	unknown	.131	.132	17.2
jayhorn-recursive/SatFibonacci01.yml | true	TIMEOUT	15.3	14.9	605	unknown	.137	.137	17.1
jayhorn-recursive/SatFibonacci02.yml | true	true	.782	.362	48.4	unknown	.131	.131	17.1
jayhorn-recursive/SatFibonacci03.yml | true	TIMEOUT	15.3	14.9	664	unknown	.131	.132	17.2
jayhorn-recursive/SatGcd.yml | true	TIMEOUT	15.3	14.8	68.1	unknown	.135	.135	17.1
jayhorn-recursive/SatHanoi01.yml | true	TIMEOUT	15.3	14.8	721	unknown	.134	.134	17.2
jayhorn-recursive/SatMccarthy91.yml | true	TIMEOUT	15.3	14.9	725	unknown	.131	.131	17.1
jayhorn-recursive/SatMultCommutative01.yml	TIMEOUT	15.3	14.8	675	unknown	.132	.132	17.2
jayhorn-recursive/SatPrimes01.yml | true	TIMEOUT	15.3	14.9	1800	unknown	.137	.137	17.1
jayhorn-recursive/UnsatAckermann01.yml | fals	false	3.97	3.56	88.9	false	.918	.503	58.8
jayhorn-recursive/UnsatAddition01.yml | false	false	.933	.564	51.5	false	.809	.448	56.0
jayhorn-recursive/UnsatAddition02.yml | false	TIMEOUT	15.3	14.9	487	unknown	.136	.137	17.1
jayhorn-recursive/UnsatEvenOdd01.yml | false	false	.908	.540	51.2	false	.993	.486	66.2
jayhorn-recursive/UnsatFibonacci01.yml | false	false	1.29	.938	54.4	false	.877	.511	56.3
jayhorn-recursive/UnsatFibonacci02.yml | false	false	4.62	4.18	303	false	.849	.471	57.3
jayhorn-recursive/UnsatMccarthy91.yml | false	false	1.03	.539	51.2	false	.899	.465	58.8
jbmc-regression/ArithmeticException1.yml | fal	false	.971	.549	51.3	false	.838	.429	58.8
jbmc-regression/ArithmeticException5.yml | tru	true	.722	.373	39.3	unknown	.132	.133	17.3
jbmc-regression/ArithmeticException6.yml | fal	false	.997	.600	51.1	false	.924	.448	58.9
jbmc-regression/ArrayIndexOutOfBoundsExcep	false	.984	.617	51.2	false	.785	.427	55.5
jbmc-regression/ArrayIndexOutOfBoundsExcep	false	.925	.549	51.2	false	.897	.449	67.1
jbmc-regression/ArrayIndexOutOfBoundsExcep	false	.919	.513	51.3	false	.788	.430	55.9
jbmc-regression/BufferedReaderReadLine.yml	false	1.02	.564	51.4	unknown	.789	.434	56.7
jbmc-regression/CharSequenceBug.yml | false	false	.875	.534	51.6	unknown	.909	.456	58.9
jbmc-regression/CharSequenceToString.yml | tr	true	1.36	.911	51.6	unknown	.138	.138	18.1
jbmc-regression/ClassCastException1.yml | false	false	.883	.505	51.7	false	.753	.419	55.1
jbmc-regression/ClassCastException2.yml | true	true	.821	.385	49.1	unknown	.135	.135	17.3
jbmc-regression/ClassCastException3.yml | false	false	.886	.511	50.9	false	.849	.478	58.0
jbmc-regression/Class_method1.yml | true	true	.686	.349	39.2	unknown	.142	.142	17.0
jbmc-regression/Inheritance1.yml | true	true	.783	.387	48.8	unknown	.135	.135	17.1
jbmc-regression/NegativeArraySizeException1.	false	.861	.505	51.2	false	.810	.444	57.9
jbmc-regression/NegativeArraySizeException2.	false	.899	.536	51.0	false	.797	.439	58.1
jbmc-regression/NullPointerException1.yml | tr	true	.704	.365	39.0	unknown	.133	.133	17.2
jbmc-regression/NullPointerException2.yml | fa	false	.895	.534	51.3	false	.860	.438	65.9
jbmc-regression/NullPointerException3.yml | fa	false	.965	.539	51.0	false	.805	.456	58.0
jbmc-regression/NullPointerException4.yml | fa	false	.936	.511	51.4	false	.828	.447	57.9
jbmc-regression/RegexMatches01.yml | true	true	.778	.389	39.8	unknown	.135	.136	17.2
jbmc-regression/RegexMatches02.yml | false	unknown	.956	.594	53.6	unknown	9.14	8.34	2030
jbmc-regression/RegexSubstitution01.yml | true	true	.898	.363	42.8	unknown	.136	.136	17.2
jbmc-regression/RegexSubstitution02.yml | false	unknown	1.02	.614	52.8	unknown	.979	.478	73.2
jbmc-regression/RegexSubstitution03.yml | true	true	.708	.335	40.3	unknown	.133	.134	17.1
jbmc-regression/StaticCharMethods01.yml | true	true	.820	.336	39.9	unknown	.141	.141	17.1
jbmc-regression/StaticCharMethods02.yml | fals	false	1.00	.610	52.4	unknown	.892	.481	66.1
jbmc-regression/StaticCharMethods03.yml | fals	false	.959	.588	52.6	unknown	.904	.447	59.7
jbmc-regression/StaticCharMethods04.yml | fals	false	.973	.575	51.8	false	.994	.564	68.8
jbmc-regression/StaticCharMethods05.yml | fals	false	1.09	.670	52.9	unknown	5.34	4.55	1910
jbmc-regression/StaticCharMethods06.yml | true	true	1.59	1.18	52.8	unknown	.164	.165	19.7

jbmc-regression/StringBuilderAppend01.yml | t	true	.769	.386	41.3	unknown	.134	.134	17.2
jbmc-regression/StringBuilderAppend02.yml | f	TIMEOUT	15.4	14.9	2820	unknown	.133	.134	17.2
jbmc-regression/StringBuilderCapLen01.yml | t	true	.698	.334	40.5	unknown	.134	.134	17.2
jbmc-regression/StringBuilderCapLen02.yml | fa	false	.922	.554	51.7	unknown	.907	.493	59.0
jbmc-regression/StringBuilderCapLen03.yml | fa	false	1.00	.544	51.5	unknown	.912	.496	59.4
jbmc-regression/StringBuilderCapLen04.yml | fa	unknown	.887	.497	51.5	unknown	5.71	5.11	2180
jbmc-regression/StringBuilderChars01.yml | true	true	.739	.364	41.2	unknown	.135	.136	17.2
jbmc-regression/StringBuilderChars02.yml | fals	false	.893	.508	51.5	unknown	.902	.473	59.7
jbmc-regression/StringBuilderChars03.yml | fals	false	.932	.547	51.2	unknown	.985	.502	69.8
jbmc-regression/StringBuilderChars04.yml | fals	false	1.02	.624	51.9	unknown	10.5	9.92	2210
jbmc-regression/StringBuilderChars05.yml | fals	false	1.01	.580	52.1	unknown	.824	.472	55.8
jbmc-regression/StringBuilderChars06.yml | fals	false	1.08	.666	64.2	unknown	.940	.497	59.2
jbmc-regression/StringBuilderConstructors01.y	true	1.30	.934	51.8	unknown	.139	.139	18.3
jbmc-regression/StringBuilderConstructors02.y	false	.947	.565	51.5	unknown	4.93	4.41	1880
jbmc-regression/StringBuilderInsertDelete01.y	true	.961	.413	51.1	unknown	.137	.138	17.2
jbmc-regression/StringBuilderInsertDelete02.y	unknown	2.28	1.87	231	unknown	10.8	10.1	2250
jbmc-regression/StringBuilderInsertDelete03.y	unknown	2.32	1.84	241	unknown	.862	.477	57.5
jbmc-regression/StringCompare01.yml | true	true	.855	.356	47.5	unknown	.150	.151	17.2
jbmc-regression/StringCompare02.yml | false	unknown	.850	.513	51.5	unknown	.818	.447	55.2
jbmc-regression/StringCompare03.yml | false	unknown	1.01	.525	51.5	unknown	3.19	2.67	862
jbmc-regression/StringCompare04.yml | false	false	1.01	.567	51.6	unknown	5.51	4.85	2060
jbmc-regression/StringCompare05.yml | false	false	.919	.538	51.7	unknown	4.00	3.52	1470
jbmc-regression/StringConcatenation01.yml | t	true	3.41	2.95	66.5	unknown	.143	.143	19.3
jbmc-regression/StringConcatenation02.yml | f	false	.983	.591	51.5	unknown	.957	.508	59.1
jbmc-regression/StringConcatenation03.yml | f	false	1.08	.680	57.3	unknown	3.04	2.59	1020
jbmc-regression/StringConcatenation04.yml | f	false	.921	.552	51.4	unknown	.956	.503	69.7
jbmc-regression/StringConstructors01.yml | true	true	.776	.369	48.9	unknown	.135	.135	17.2
jbmc-regression/StringConstructors02.yml | fals	false	.927	.547	51.7	unknown	3.12	2.61	1010
jbmc-regression/StringConstructors03.yml | fals	false	1.04	.627	51.6	unknown	11.0	10.4	2290
jbmc-regression/StringConstructors04.yml | fals	false	1.19	.822	52.4	unknown	.903	.504	59.8
jbmc-regression/StringConstructors05.yml | fals	false	1.61	1.22	56.6	unknown	.875	.463	60.1
jbmc-regression/StringContains01.yml | false	false	.985	.629	65.6	unknown	5.21	4.60	1940
jbmc-regression/StringContains02.yml | false	false	.954	.548	51.2	false	3.47	2.92	980
jbmc-regression/StringIndexMethods01.yml | tru	true	.758	.334	46.8	unknown	.135	.135	17.2
jbmc-regression/StringIndexMethods02.yml | fal	false	.947	.514	51.7	unknown	.845	.446	58.7
jbmc-regression/StringIndexMethods03.yml | fal	false	1.06	.614	51.3	unknown	5.51	4.92	2110
jbmc-regression/StringIndexMethods04.yml | fal	false	.990	.560	51.6	unknown	.948	.476	68.8
jbmc-regression/StringIndexMethods05.yml | fal	false	.882	.513	51.2	false	2.65	2.15	648
jbmc-regression/StringMiscellaneous01.yml | tru	true	.765	.381	41.0	unknown	.137	.137	17.2
jbmc-regression/StringMiscellaneous02.yml | fal	false	.902	.508	50.9	unknown	.978	.499	67.6
jbmc-regression/StringMiscellaneous03.yml | fal	false	1.01	.656	52.6	unknown	.965	.481	68.1
jbmc-regression/StringMiscellaneous04.yml | tru	true	.787	.340	48.7	unknown	.147	.147	17.2
jbmc-regression/StringStartEnd01.yml | true	true	.709	.359	39.5	unknown	.131	.131	17.2
jbmc-regression/StringStartEnd02.yml | false	false	1.28	.831	51.5	unknown	1.02	.519	69.7
jbmc-regression/StringStartEnd03.yml | false	false	1.20	.812	51.7	unknown	9.07	8.39	2020
jbmc-regression/StringValueOf01.yml | true	true	.779	.355	48.5	unknown	.131	.131	17.1
jbmc-regression/StringValueOf02.yml | false	false	1.24	.762	55.2	unknown	2.96	2.45	960
jbmc-regression/StringValueOf03.yml | false	false	1.14	.763	54.5	unknown	5.61	5.06	2160
jbmc-regression/StringValueOf04.yml | false	false	.873	.508	51.3	false	.846	.461	58.8
jbmc-regression/StringValueOf05.yml | false	false	1.03	.587	52.0	unknown	.935	.510	59.5
jbmc-regression/StringValueOf06.yml | false	false	.944	.571	52.2	false	.887	.490	58.7
jbmc-regression/StringValueOf07.yml | false	false	1.45	1.06	56.8	false	1.06	.515	72.5
jbmc-regression/StringValueOf08.yml | false	false	7.48	7.06	361	unknown	5.19	4.61	1940
jbmc-regression/StringValueOf09.yml | false	unknown	4.21	3.79	355	unknown	5.35	4.85	2030
jbmc-regression/StringValueOf10.yml | false	false	.948	.556	51.6	unknown	.888	.451	59.4
jbmc-regression/SubString01.yml | true	true	.698	.347	39.5	unknown	.134	.134	17.2
jbmc-regression/SubString02.yml | false	false	.953	.570	52.2	unknown	10.9	10.3	2280
jbmc-regression/SubString03.yml | false	false	.960	.558	52.2	unknown	.924	.468	58.9

jbmc-regression/TokenTest01.yml	true	true	.845	.381	47.0	unknown	.149	.150	17.1
jbmc-regression/TokenTest02.yml	false	TIMEOUT	15.3	14.9	2260	unknown	.134	.134	17.1
jbmc-regression/Validate01.yml	true	true	.731	.368	40.3	unknown	.135	.135	17.2
jbmc-regression/Validate02.yml	false	unknown	1.08	.634	51.9	unknown	1.04	.522	72.7
jbmc-regression/aastore_aaload1.yml	true	TIMEOUT	15.2	14.9	374	unknown	.135	.135	17.2
jbmc-regression/array1.yml	true	TIMEOUT	15.3	14.9	492	unknown	.137	.137	17.2
jbmc-regression/array2.yml	true	true	1.30	.951	51.3	unknown	.138	.138	18.6
jbmc-regression/arraylength1.yml	true	true	1.36	.945	51.7	unknown	.145	.146	18.0
jbmc-regression/arrayread1.yml	true	true	1.27	.903	51.8	unknown	.135	.135	17.9
jbmc-regression/assert1.yml	true	true	1.65	1.26	53.7	unknown	.137	.137	18.5
jbmc-regression/assert2.yml	false	false	.985	.612	53.2	false	.875	.462	58.7
jbmc-regression/assert3.yml	false	false	1.03	.551	51.2	false	.851	.467	58.5
jbmc-regression/assert4.yml	false	false	1.15	.637	53.7	false	.909	.513	58.0
jbmc-regression/assert5.yml	true	true	1.66	1.23	53.2	unknown	.134	.134	18.0
jbmc-regression/assert6.yml	true	true	1.32	.906	51.7	unknown	.138	.138	17.9
jbmc-regression/astore_aload1.yml	true	true	.825	.405	41.3	unknown	.134	.134	17.1
jbmc-regression/athrow1.yml	false	false	.897	.515	51.2	false	.943	.491	58.2
jbmc-regression/basic1.yml	true	true	.721	.334	39.1	unknown	.136	.136	17.2
jbmc-regression/bitwise1.yml	true	true	1.26	.868	51.8	unknown	.136	.136	17.9
jbmc-regression/boolean1.yml	true	true	1.31	.938	50.8	unknown	.136	.136	18.0
jbmc-regression/boolean2.yml	true	true	1.28	.895	51.5	unknown	.135	.135	17.9
jbmc-regression/bug-test-gen-095.yml	false	false	1.01	.557	51.7	false	6.95	5.92	2470
jbmc-regression/bug-test-gen-119-2.yml	true	true	.728	.367	39.4	unknown	.157	.157	17.2
jbmc-regression/bug-test-gen-119.yml	true	true	1.32	.942	51.9	unknown	.138	.139	18.6
jbmc-regression/calc.yml	true	TIMEOUT	15.3	14.9	88.9	unknown	.131	.131	17.2
jbmc-regression/cast1.yml	true	true	1.42	1.03	52.1	unknown	.137	.137	19.2
jbmc-regression/catch1.yml	true	true	.781	.366	49.2	unknown	.130	.130	17.2
jbmc-regression/char1.yml	true	true	1.38	1.01	51.9	unknown	.137	.138	18.6
jbmc-regression/charArray.yml	true	true	2.35	1.99	96.7	unknown	.145	.145	21.7
jbmc-regression/classtest1.yml	true	true	.690	.364	39.7	unknown	.136	.136	17.2
jbmc-regression/const1.yml	true	true	.726	.356	39.2	unknown	.133	.134	17.2
jbmc-regression/constructor1.yml	true	true	.779	.394	39.6	unknown	.133	.133	17.2
jbmc-regression/enum1.yml	true	true	.737	.372	40.1	unknown	.133	.133	17.2
jbmc-regression/exceptions1.yml	false	false	.963	.568	51.4	false	.793	.470	55.2
jbmc-regression/exceptions10.yml	false	false	.947	.547	51.4	false	.858	.474	59.0
jbmc-regression/exceptions11.yml	false	false	.928	.537	51.5	false	.790	.478	55.6
jbmc-regression/exceptions12.yml	false	false	.962	.575	51.9	false	.824	.488	55.8
jbmc-regression/exceptions13.yml	false	false	.947	.560	51.7	false	.971	.518	66.9
jbmc-regression/exceptions14.yml	true	true	.736	.363	39.5	unknown	.134	.134	17.2
jbmc-regression/exceptions15.yml	true	true	.670	.324	39.6	unknown	.145	.145	17.2
jbmc-regression/exceptions16.yml	false	false	.973	.586	51.4	false	.823	.493	55.7
jbmc-regression/exceptions18.yml	true	true	.811	.372	39.5	unknown	.145	.146	17.2
jbmc-regression/exceptions2.yml	false	false	.900	.524	51.4	false	.890	.462	58.0
jbmc-regression/exceptions3.yml	false	false	.950	.534	51.3	false	.902	.478	67.0
jbmc-regression/exceptions4.yml	true	true	.713	.361	40.0	unknown	.130	.130	17.2
jbmc-regression/exceptions5.yml	true	true	.775	.380	40.4	unknown	.134	.134	17.1
jbmc-regression/exceptions6.yml	false	false	.896	.514	51.0	false	.913	.474	64.8
jbmc-regression/exceptions7.yml	false	false	.968	.528	51.7	false	.793	.461	55.2
jbmc-regression/exceptions8.yml	false	false	.987	.541	51.1	false	.811	.469	55.8
jbmc-regression/exceptions9.yml	true	true	.726	.367	41.6	unknown	.154	.155	17.1
jbmc-regression/fcmpx_dcmpx1.yml	true	true	.720	.364	39.5	unknown	.131	.132	17.2
jbmc-regression/iarith1.yml	true	true	.657	.318	39.7	unknown	.132	.132	17.2
jbmc-regression/iarith2.yml	true	true	.749	.354	41.1	unknown	.132	.132	17.1
jbmc-regression/if_acmp1.yml	true	true	.711	.352	40.1	unknown	.132	.132	17.2
jbmc-regression/if_expr1.yml	true	true	1.28	.907	51.5	unknown	.138	.138	18.3
jbmc-regression/if_icmp1.yml	true	true	1.27	.905	51.7	unknown	.135	.135	18.5
jbmc-regression/ifxx1.yml	true	true	.671	.327	40.0	unknown	.152	.152	17.2
jbmc-regression/instanceof1.yml	true	true	.653	.317	39.6	unknown	.133	.133	17.2

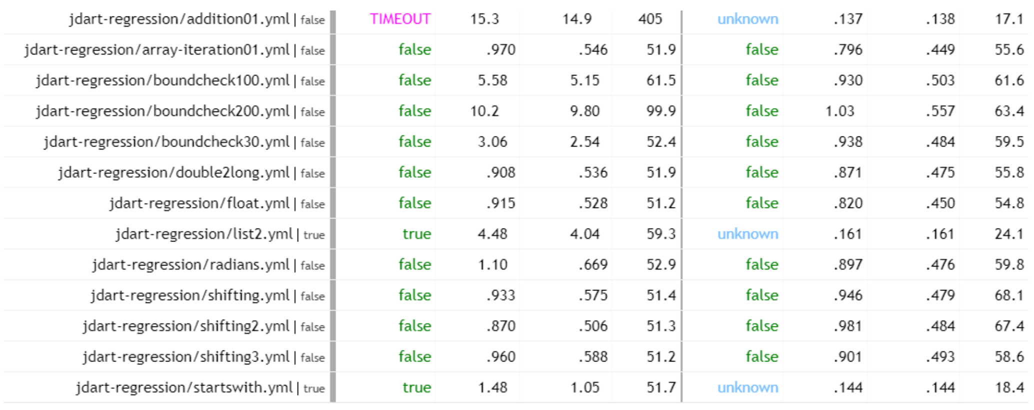
jbmc-regression/instanceof2.yml | true	true	.715	.362	39.4	unknown	.132	.132	17.1
jbmc-regression/instanceof3.yml | true	true	.681	.332	39.1	unknown	.134	.134	17.2
jbmc-regression/instanceof4.yml | true	true	.707	.398	39.1	unknown	.133	.133	17.2
jbmc-regression/instanceof5.yml | true	true	.760	.337	39.8	unknown	.133	.133	17.2
jbmc-regression/instanceof6.yml | true	true	.703	.351	39.9	unknown	.149	.149	17.2
jbmc-regression/instanceof7.yml | true	true	.761	.366	49.5	unknown	.133	.133	17.2
jbmc-regression/instanceof8.yml | true	true	.660	.316	39.8	unknown	.131	.131	17.2
jbmc-regression/interface1.yml | false	false	.863	.502	51.6	false	.786	.481	55.1
jbmc-regression/java_append_char.yml | false	false	1.36	.976	57.7	false	.976	.499	70.0
jbmc-regression/lazyloading4.yml | true	true	.808	.378	49.7	unknown	.134	.134	17.2
jbmc-regression/list1.yml | true	TIMEOUT	15.3	14.8	79.6	unknown	.133	.134	17.2
jbmc-regression/long1.yml | true	true	.786	.374	49.1	unknown	.133	.133	17.2
jbmc-regression/lookupswitch1.yml | true	true	1.28	.897	51.5	unknown	.137	.137	18.4
jbmc-regression/multinewarray.yml | true	true	.678	.316	39.3	unknown	.131	.131	17.2
jbmc-regression/overloading1.yml | true	true	.839	.394	40.0	unknown	.130	.131	17.1
jbmc-regression/package1.yml | true	true	.741	.318	39.6	unknown	.132	.133	17.1
jbmc-regression/putfield_getfield1.yml | true	true	.659	.314	38.7	unknown	.131	.132	17.2
jbmc-regression/putstatic_getstatic1.yml | true	true	.758	.343	49.2	unknown	.131	.132	17.2
jbmc-regression/recursion2.yml | true	true	.787	.382	48.1	unknown	.130	.131	17.2
jbmc-regression/return1.yml | false	false	.943	.525	51.3	false	.780	.425	58.2
jbmc-regression/return2.yml | false	false	1.07	.607	51.6	false	.974	.506	67.8
jbmc-regression/store_load1.yml | true	true	.795	.378	48.5	unknown	.135	.135	17.2
jbmc-regression/swap1.yml | true	true	.727	.371	39.3	unknown	.134	.135	17.2
jbmc-regression/synchronized.yml | true	true	.668	.305	39.8	unknown	.140	.140	17.1
jbmc-regression/tableswitch1.yml | true	true	1.40	.961	51.7	unknown	.144	.144	18.5
jbmc-regression/uninitialised1.yml | true	true	.732	.349	48.0	unknown	.174	.174	17.1
jbmc-regression/virtual1.yml | true	true	.791	.368	48.8	unknown	.139	.139	17.1
jbmc-regression/virtual2.yml | false	false	.976	.554	51.0	false	.934	.484	66.6
jbmc-regression/virtual4.yml | true	true	.799	.373	39.4	unknown	.145	.145	17.1
jbmc-regression/virtual_function_unwinding.y	true	.805	.390	48.6	unknown	.137	.137	17.2
jpf-regression/ExDarko_false.yml | false	false	1.10	.610	52.0	false	1.04	.541	58.3
jpf-regression/ExDarko_true.yml | true	true	1.40	1.03	52.6	unknown	.139	.140	19.0
jpf-regression/ExException_false.yml | false	false	1.00	.550	51.6	false	.895	.471	58.6
jpf-regression/ExException_true.yml | true	true	1.33	.937	51.4	unknown	.137	.137	18.0
jpf-regression/ExGenSymExe_false.yml | false	false	.924	.533	51.1	false	.926	.497	65.6
jpf-regression/ExGenSymExe_true.yml | true	true	1.26	.899	51.8	unknown	.145	.145	18.3
jpf-regression/ExLazy_false.yml | false	false	.936	.498	51.7	false	1.11	.516	68.9
jpf-regression/ExLazy_true.yml | true	true	1.36	.936	51.8	unknown	.145	.146	18.1
jpf-regression/ExMIT_false.yml | false	false	.893	.512	51.0	unknown	.791	.446	55.7
jpf-regression/ExMIT_true.yml | true	true	1.30	.883	51.3	unknown	.142	.142	17.9
jpf-regression/ExSymExe10_false.yml | false	false	.900	.525	51.5	false	.923	.480	66.9
jpf-regression/ExSymExe10_true.yml | true	true	1.41	.931	51.7	unknown	.154	.154	17.9
jpf-regression/ExSymExe11_false.yml | false	false	.961	.515	51.2	false	.995	.519	65.9
jpf-regression/ExSymExe11_true.yml | true	true	1.38	.908	51.4	unknown	.139	.139	17.9
jpf-regression/ExSymExe12_false.yml | false	false	.963	.555	51.7	false	.915	.513	56.2
jpf-regression/ExSymExe12_true.yml | true	true	1.43	.960	51.6	unknown	.147	.147	18.1
jpf-regression/ExSymExe13_false.yml | false	false	.958	.557	51.4	false	.859	.485	55.9
jpf-regression/ExSymExe13_true.yml | true	true	1.47	.997	52.1	unknown	.141	.141	18.1
jpf-regression/ExSymExe14_false.yml | false	false	1.01	.549	51.3	unknown	.812	.466	56.1
jpf-regression/ExSymExe14_true.yml | true	true	1.31	.939	52.2	unknown	.145	.145	18.2
jpf-regression/ExSymExe15_false.yml | false	false	.978	.536	51.6	false	.927	.532	56.5
jpf-regression/ExSymExe15_true.yml | true	true	1.46	1.00	51.5	unknown	.140	.141	18.1
jpf-regression/ExSymExe16_false.yml | false	false	.925	.535	51.7	false	.784	.434	54.8
jpf-regression/ExSymExe16_true.yml | true	true	.731	.351	39.4	unknown	.139	.140	17.2
jpf-regression/ExSymExe17_false.yml | false	false	.928	.533	51.4	false	.734	.420	55.2
jpf-regression/ExSymExe17_true.yml | true	true	.675	.313	40.1	unknown	.140	.141	17.1
jpf-regression/ExSymExe18_false.yml | false	false	.911	.539	51.5	false	.895	.474	57.1
jpf-regression/ExSymExe18_true.yml | true	true	.803	.377	49.1	unknown	.135	.136	17.2

jpf-regression/ExSymExe19_false.yml	false	false	.977	.584	51.8	false	.857	.494	55.6
jpf-regression/ExSymExe19_true.yml	true	true	1.37	1.00	51.7	unknown	.143	.143	18.0
jpf-regression/ExSymExe1_false.yml	false	false	.914	.546	51.9	true	.927	.496	56.7
jpf-regression/ExSymExe1_true.yml	true	true	.834	.387	49.1	unknown	.143	.144	17.3
jpf-regression/ExSymExe20_false.yml	false	false	.975	.576	51.1	false	.832	.472	56.5
jpf-regression/ExSymExe20_true.yml	true	true	.673	.333	39.5	unknown	.139	.139	17.2
jpf-regression/ExSymExe21_false.yml	false	false	.946	.559	51.5	false	.928	.462	55.5
jpf-regression/ExSymExe21_true.yml	true	true	.715	.347	39.9	unknown	.144	.144	17.2
jpf-regression/ExSymExe25_false.yml	false	false	.944	.532	51.7	false	.835	.474	56.4
jpf-regression/ExSymExe25_true.yml	true	true	.736	.388	39.8	unknown	.139	.139	17.2
jpf-regression/ExSymExe26_false.yml	false	false	1.02	.553	51.6	false	.805	.457	56.8
jpf-regression/ExSymExe26_true.yml	true	true	.749	.334	39.5	unknown	.139	.139	17.2
jpf-regression/ExSymExe27_false.yml	false	false	1.11	.594	51.8	false	.935	.498	67.2
jpf-regression/ExSymExe27_true.yml	true	true	.735	.345	39.5	unknown	.141	.141	17.1
jpf-regression/ExSymExe28_false.yml	false	false	.999	.582	51.4	false	.877	.461	66.5
jpf-regression/ExSymExe28_true.yml	true	true	.824	.371	38.9	unknown	.140	.141	17.3
jpf-regression/ExSymExe29_false.yml	false	false	.952	.562	51.5	false	.943	.483	68.0
jpf-regression/ExSymExe29_true.yml	true	true	.716	.351	39.6	unknown	.144	.145	17.2
jpf-regression/ExSymExe2_false.yml	false	false	1.11	.586	51.7	false	.973	.495	69.0
jpf-regression/ExSymExe2_true.yml	true	true	.759	.333	39.1	unknown	.139	.140	17.2
jpf-regression/ExSymExe3_false.yml	false	false	.939	.554	51.5	false	.847	.487	56.5
jpf-regression/ExSymExe3_true.yml	true	true	.699	.328	39.5	unknown	.143	.144	17.1
jpf-regression/ExSymExe4_false.yml	false	false	.901	.534	51.6	false	.829	.480	56.4
jpf-regression/ExSymExe4_true.yml	true	true	.788	.340	49.0	unknown	.141	.142	17.1
jpf-regression/ExSymExe5_false.yml	false	false	.912	.522	51.4	false	.823	.469	56.2
jpf-regression/ExSymExe5_true.yml	true	true	.706	.338	39.8	unknown	.143	.144	17.2
jpf-regression/ExSymExe6_false.yml	false	false	1.06	.565	51.3	false	.911	.499	56.3
jpf-regression/ExSymExe6_true.yml	true	true	.717	.352	39.6	unknown	.141	.142	17.2
jpf-regression/ExSymExe7_false.yml	false	false	.954	.547	51.7	false	.921	.531	56.2
jpf-regression/ExSymExe7_true.yml	true	true	1.33	.961	51.9	unknown	.146	.146	18.1
jpf-regression/ExSymExe8_false.yml	false	false	.881	.520	51.3	false	.918	.508	56.6
jpf-regression/ExSymExe8_true.yml	true	true	.696	.333	39.8	unknown	.151	.151	17.2
jpf-regression/ExSymExe9_false.yml	false	false	.931	.534	51.3	false	.891	.483	55.5
jpf-regression/ExSymExe9_true.yml	true	true	.694	.322	40.3	unknown	.143	.143	17.2
jpf-regression/ExSymExeArrays_false.yml	false	false	.993	.565	51.3	false	.820	.448	55.4
jpf-regression/ExSymExeArrays_true.yml	true	true	.676	.311	40.2	unknown	.137	.138	17.2
jpf-regression/ExSymExeBool_false.yml	false	false	.892	.521	51.3	false	.866	.490	64.1
jpf-regression/ExSymExeBool_true.yml	true	true	.714	.360	39.6	unknown	.140	.141	17.1
jpf-regression/ExSymExeComplexMath_false.yı		false	1.01	.571	52.4	false	.860	.544	57.0
jpf-regression/ExSymExeComplexMath_true.yn		true	.737	.344	40.4	unknown	.136	.137	17.2
jpf-regression/ExSymExeD2I_false.yml	false	false	.988	.547	51.1	false	.906	.472	65.5
jpf-regression/ExSymExeD2I_true.yml	true	true	1.41	1.02	51.7	unknown	.142	.142	18.1
jpf-regression/ExSymExeD2L_false.yml	false	false	1.04	.576	51.5	false	1.09	.533	70.4
jpf-regression/ExSymExeD2L_true.yml	true	true	1.46	1.01	51.7	unknown	.155	.156	18.2
jpf-regression/ExSymExeF2I_false.yml	false	false	.902	.533	51.6	false	.823	.443	58.4
jpf-regression/ExSymExeF2I_true.yml	true	true	1.32	.952	51.6	unknown	.149	.149	18.0
jpf-regression/ExSymExeF2L_false.yml	false	false	.913	.539	51.6	false	.887	.480	59.1
jpf-regression/ExSymExeF2L_true.yml	true	true	1.32	.937	51.6	unknown	.143	.143	18.1
jpf-regression/ExSymExeFNEG_false.yml	false	false	.956	.517	51.4	false	.897	.494	59.1
jpf-regression/ExSymExeFNEG_true.yml	true	true	1.31	.932	51.3	unknown	.140	.140	18.0
jpf-regression/ExSymExeGetStatic_false.yml	f	false	1.02	.560	51.7	false	.764	.433	55.7
jpf-regression/ExSymExeGetStatic_true.yml	tr	true	.843	.369	48.4	unknown	.136	.136	17.2
jpf-regression/ExSymExeI2D_false.yml	false	false	.962	.514	51.6	false	.922	.504	59.0
jpf-regression/ExSymExeI2D_true.yml	true	true	1.34	.947	51.9	unknown	.139	.139	18.1
jpf-regression/ExSymExeI2F_false.yml	false	false	.884	.502	50.9	false	.845	.443	58.4
jpf-regression/ExSymExeI2F_true.yml	true	true	1.37	.948	51.6	unknown	.140	.140	17.9
jpf-regression/ExSymExeLCMP_false.yml	false	false	.916	.512	51.5	false	.929	.480	66.9
jpf-regression/ExSymExeLCMP_true.yml	true	true	1.31	.941	51.5	unknown	.151	.152	18.1

jpf-regression/ExSymExeLongBytecodes_false.	false	.962	.530	51.4	false	.951	.509	65.9
jpf-regression/ExSymExeLongBytecodes_true.y	true	1.39	1.01	52.1	unknown	.149	.150	18.1
jpf-regression/ExSymExeResearch_false.yml | fa	false	.994	.546	51.2	false	.827	.478	55.8
jpf-regression/ExSymExeResearch_true.yml | tr	true	1.37	.914	51.7	unknown	.143	.144	18.1
jpf-regression/ExSymExeSimple_false.yml | false	false	1.01	.547	51.5	false	1.06	.515	68.9
jpf-regression/ExSymExeSimple_true.yml | true	true	1.30	.893	52.0	unknown	.156	.156	18.1
jpf-regression/ExSymExeSuzette_false.yml | fal	false	.922	.544	51.9	false	.901	.507	55.4
jpf-regression/ExSymExeSuzette_true.yml | true	true	1.30	.946	51.9	unknown	.150	.150	18.0
jpf-regression/ExSymExeSwitch_false.yml | false	false	.885	.501	51.3	false	.846	.456	59.7
jpf-regression/ExSymExeSwitch_true.yml | true	true	1.34	.892	51.2	unknown	.147	.147	18.0
jpf-regression/ExSymExeTestAssignments_false	false	.901	.534	51.5	false	.901	.481	55.8
jpf-regression/ExSymExeTestAssignments_true	true	1.34	.949	51.4	unknown	.141	.141	17.8
jpf-regression/ExSymExeTestClassFields_false.	false	.920	.549	51.5	true	.920	.471	67.7
jpf-regression/ExSymExeTestClassFields_true.y	true	1.22	.865	51.7	unknown	.147	.147	17.7
jpf-regression/ExSymExe_false.yml | false	false	1.02	.579	52.2	false	.945	.500	68.6
jpf-regression/ExSymExe_true.yml | true	true	.919	.408	58.6	unknown	.141	.141	17.1
jpf-regression/TestLazy_false.yml | false	false	.971	.602	51.9	false	.893	.469	59.1
jpf-regression/TestLazy_true.yml | true	true	1.29	.926	52.4	unknown	.141	.142	18.3
java-ranger-regression/TCAS_prop1.yml | false	false	1.04	.622	53.0	unknown	.844	.462	58.0
java-ranger-regression/TCAS_prop2.yml | true	true	1.53	1.17	53.2	unknown	.148	.148	20.5
java-ranger-regression/TCAS_prop3.yml | true	true	1.64	1.25	53.3	unknown	.157	.157	20.5
java-ranger-regression/apachecli_eqchk.yml | t	TIMEOUT	15.8	14.0	141	unknown	.142	.142	17.2
java-ranger-regression/schedule_eqchk.yml | tr	TIMEOUT	15.3	14.8	893	unknown	.176	.177	17.1
java-ranger-regression/WBS/WBS_prop1.yml | f	false	1.03	.611	52.5	false	1.04	.522	69.8
java-ranger-regression/WBS/WBS_prop2.yml | t	true	1.43	1.04	52.8	unknown	.154	.154	19.8
java-ranger-regression/WBS/WBS_prop3.yml | f	false	1.01	.608	52.6	false	.958	.569	57.7
java-ranger-regression/WBS/WBS_prop4.yml | f	false	1.07	.647	53.0	false	.888	.508	58.2
java-ranger-regression/alarm/Alarm_prop1.ym	true	4.45	3.33	105	unknown	.187	.187	27.2
java-ranger-regression/alarm/Alarm_prop10.y	true	4.45	3.40	105	unknown	.170	.170	27.0
java-ranger-regression/alarm/Alarm_prop2.ym	true	4.70	3.45	105	unknown	.171	.171	27.1
java-ranger-regression/alarm/Alarm_prop3.ym	true	4.68	3.46	105	unknown	.172	.172	27.2
java-ranger-regression/alarm/Alarm_prop4.ym	true	4.31	3.39	105	unknown	.172	.172	27.1
java-ranger-regression/alarm/Alarm_prop5.ym	true	4.43	3.39	105	unknown	.170	.170	27.1
java-ranger-regression/alarm/Alarm_prop6.ym	true	4.41	3.44	106	unknown	.171	.171	27.2
java-ranger-regression/alarm/Alarm_prop8.ym	true	4.70	3.51	105	unknown	.169	.170	27.2
java-ranger-regression/alarm/Alarm_prop9.ym	true	4.66	3.43	105	unknown	.169	.170	27.1
java-ranger-regression/infusion/Infusion_prop1	true	2.83	1.90	70.3	unknown	.151	.151	21.3
java-ranger-regression/infusion/Infusion_prop1	true	2.83	1.91	70.0	unknown	.150	.150	21.2
java-ranger-regression/infusion/Infusion_prop1	true	2.67	1.85	65.8	unknown	.156	.156	21.3
java-ranger-regression/infusion/Infusion_prop1	true	2.58	1.80	64.7	unknown	.164	.164	21.1
java-ranger-regression/infusion/Infusion_prop1	true	2.68	1.87	64.0	unknown	.156	.157	21.3
java-ranger-regression/infusion/Infusion_prop2	true	2.57	1.80	64.9	unknown	.149	.149	21.2
java-ranger-regression/infusion/Infusion_prop3	true	2.66	1.85	63.4	unknown	.149	.149	21.2
java-ranger-regression/infusion/Infusion_prop4	true	2.78	1.88	65.9	unknown	.147	.147	21.2
java-ranger-regression/infusion/Infusion_prop5	true	2.76	1.88	64.5	unknown	.149	.149	21.3
java-ranger-regression/infusion/Infusion_prop6	true	2.63	1.85	65.0	unknown	.148	.148	21.3
java-ranger-regression/infusion/Infusion_prop7	true	2.58	1.87	62.9	unknown	.149	.149	21.3
java-ranger-regression/infusion/Infusion_prop8	true	2.62	1.86	62.9	unknown	.150	.150	21.4
java-ranger-regression/infusion/Infusion_prop9	true	2.67	1.91	65.3	unknown	.162	.162	21.3
java-ranger-regression/siena_eqchk/siena_pro	TIMEOUT	15.6	14.4	296	unknown	.136	.136	17.2
java-ranger-regression/siena_eqchk/siena_pro	TIMEOUT	15.6	14.4	293	unknown	.137	.138	17.2
java-ranger-regression/replace5_eqchk/replac	TIMEOUT	15.4	14.7	1290	unknown	.135	.136	17.2
java-ranger-regression/replace5_eqchk/replac	TIMEOUT	15.4	14.7	1300	unknown	.138	.138	17.1
java-ranger-regression/nanoxml_eqchk/nanox	TIMEOUT	15.7	14.2	1130	unknown	.140	.141	17.1
java-ranger-regression/nanoxml_eqchk/nanox	TIMEOUT	15.6	14.3	1140	unknown	.137	.138	17.1
java-ranger-regression/nanoxml_eqchk/nanox	TIMEOUT	15.6	14.3	1130	unknown	.138	.138	17.2
java-ranger-regression/printtokens_eqchk/prir	TIMEOUT	15.3	14.8	504	unknown	.134	.134	17.1
java-ranger-regression/printtokens_eqchk/prir	TIMEOUT	15.3	14.8	504	unknown	.138	.138	17.3

MinePump/spec1-5_product1.yml | false	false	1.53	1.07	55.9	false	1.01	.534	63.3
MinePump/spec1-5_product10.yml | false	false	1.61	1.10	56.2	unknown	1.09	.566	63.2
MinePump/spec1-5_product11.yml | false	false	1.79	1.21	61.1	false	1.20	.557	78.3
MinePump/spec1-5_product12.yml | false	false	1.64	1.18	56.1	false	1.01	.527	63.5
MinePump/spec1-5_product13.yml | false	false	1.58	1.15	56.1	unknown	1.17	.536	79.6
MinePump/spec1-5_product14.yml | false	false	1.60	1.15	56.8	false	1.07	.555	63.3
MinePump/spec1-5_product15.yml | false	false	1.75	1.20	58.4	false	1.09	.576	63.6
MinePump/spec1-5_product16.yml | false	false	1.57	1.14	56.2	unknown	1.04	.552	64.2
MinePump/spec1-5_product17.yml | false	false	1.89	1.20	60.5	false	1.13	.593	63.6
MinePump/spec1-5_product18.yml | false	false	1.63	1.17	56.1	false	1.13	.563	76.9
MinePump/spec1-5_product19.yml | false	false	1.65	1.22	56.4	false	1.05	.558	63.9
MinePump/spec1-5_product2.yml | false	false	1.58	1.15	56.0	false	1.11	.568	63.4
MinePump/spec1-5_product20.yml | false	false	1.59	1.16	56.9	false	1.12	.549	77.6
MinePump/spec1-5_product21.yml | false	false	1.60	1.16	56.3	false	1.07	.545	63.9
MinePump/spec1-5_product22.yml | false	false	1.61	1.16	56.6	false	1.16	.555	80.3
MinePump/spec1-5_product23.yml | false	false	1.73	1.18	56.9	false	1.17	.540	76.2
MinePump/spec1-5_product24.yml | false	false	1.73	1.20	56.4	false	.943	.526	62.1
MinePump/spec1-5_product25.yml | false	false	1.56	1.13	56.1	false	1.09	.564	64.0
MinePump/spec1-5_product26.yml | false	false	1.91	1.28	62.9	false	1.04	.544	64.4
MinePump/spec1-5_product27.yml | false	false	1.75	1.19	56.4	false	1.01	.535	63.5
MinePump/spec1-5_product28.yml | false	false	1.63	1.18	56.3	false	1.01	.541	64.5
MinePump/spec1-5_product29.yml | false	false	1.59	1.15	56.1	false	1.01	.523	64.0
MinePump/spec1-5_product3.yml | false	false	1.61	1.19	56.0	false	1.03	.534	62.9
MinePump/spec1-5_product30.yml | false	false	1.61	1.15	56.6	false	1.08	.560	64.4
MinePump/spec1-5_product31.yml | false	false	1.62	1.18	57.0	false	1.11	.535	79.0
MinePump/spec1-5_product32.yml | false	false	1.60	1.17	56.5	false	1.06	.558	64.3
MinePump/spec1-5_product33.yml | false	false	1.72	1.28	60.2	false	1.03	.541	62.3
MinePump/spec1-5_product34.yml | false	false	1.76	1.33	62.2	false	.989	.522	62.4
MinePump/spec1-5_product35.yml | false	false	1.79	1.33	64.5	false	1.03	.548	63.0
MinePump/spec1-5_product36.yml | false	false	1.78	1.35	67.7	false	1.03	.560	66.0
MinePump/spec1-5_product37.yml | false	false	1.81	1.35	64.2	false	1.09	.582	64.7
MinePump/spec1-5_product38.yml | false	false	1.99	1.39	62.8	false	.987	.525	63.1
MinePump/spec1-5_product39.yml | false	false	1.75	1.29	67.7	false	1.03	.556	65.8
MinePump/spec1-5_product4.yml | false	false	1.56	1.13	56.9	false	1.02	.545	64.2
MinePump/spec1-5_product40.yml | false	false	1.88	1.35	65.5	false	.987	.517	63.6
MinePump/spec1-5_product41.yml | false	false	1.93	1.35	64.5	unknown	1.17	.552	78.2
MinePump/spec1-5_product42.yml | false	false	2.02	1.39	66.6	false	1.07	.573	67.6
MinePump/spec1-5_product43.yml | false	false	1.80	1.35	66.2	false	1.16	.575	79.7
MinePump/spec1-5_product44.yml | false	false	1.87	1.39	67.0	false	1.27	.582	82.0
MinePump/spec1-5_product45.yml | false	false	1.79	1.33	66.4	false	1.37	.636	82.0
MinePump/spec1-5_product46.yml | false	false	1.83	1.41	66.4	false	1.10	.581	65.5
MinePump/spec1-5_product47.yml | false	false	1.82	1.37	69.5	false	1.13	.623	66.3
MinePump/spec1-5_product48.yml | false	false	1.81	1.37	68.8	false	1.08	.583	65.2
MinePump/spec1-5_product49.yml | false	false	1.82	1.32	63.7	false	.948	.526	61.3
MinePump/spec1-5_product5.yml | false	false	1.62	1.19	56.0	false	1.07	.568	63.2
MinePump/spec1-5_product50.yml | false	false	2.03	1.42	64.5	false	1.03	.553	63.4
MinePump/spec1-5_product51.yml | false	false	1.84	1.39	65.7	false	1.06	.568	61.9
MinePump/spec1-5_product52.yml | false	false	2.02	1.43	66.8	false	1.16	.555	79.7
MinePump/spec1-5_product53.yml | false	false	1.85	1.36	64.4	false	1.05	.564	62.7
MinePump/spec1-5_product54.yml | false	false	1.81	1.34	64.5	false	1.14	.560	62.2
MinePump/spec1-5_product55.yml | false	false	1.96	1.39	66.7	false	1.03	.531	64.4
MinePump/spec1-5_product56.yml | false	false	1.89	1.41	67.0	false	1.01	.537	63.3
MinePump/spec1-5_product57.yml | true	true	2.15	1.72	70.2	unknown	.195	.196	32.4
MinePump/spec1-5_product58.yml | true	true	2.38	1.80	71.2	unknown	.193	.193	32.6
MinePump/spec1-5_product59.yml | true	true	2.25	1.81	72.7	unknown	.194	.194	33.2
MinePump/spec1-5_product6.yml | false	false	1.78	1.22	60.4	false	1.03	.547	63.8
MinePump/spec1-5_product60.yml | true	true	2.28	1.85	73.7	unknown	.205	.206	33.6
MinePump/spec1-5_product61.yml | true	true	2.23	1.77	71.2	unknown	.196	.197	32.8

MinePump/spec1-5_product62.yml | true	true	2.23	1.77	72.3	unknown	.196	.196	33.1
MinePump/spec1-5_product63.yml | true	true	2.35	1.90	73.7	unknown	.201	.201	33.5
MinePump/spec1-5_product64.yml | true	true	2.34	1.90	74.0	unknown	.197	.197	33.9
MinePump/spec1-5_product7.yml | false	false	1.76	1.20	59.8	false	1.17	.545	80.8
MinePump/spec1-5_product8.yml | false	false	1.60	1.16	56.6	false	.985	.537	63.3
MinePump/spec1-5_product9.yml | false	false	1.74	1.16	56.6	false	1.18	.574	76.9
algorithms/BellmanFord-FunSat01.yml | true	TIMEOUT	15.3	14.9	148	unknown	.147	.147	17.0
algorithms/BellmanFord-FunSat02.yml | false	TIMEOUT	15.3	14.9	159	unknown	.146	.147	17.2
algorithms/BellmanFord-FunUnsat01.yml | false	false	1.13	.622	52.6	unknown	.992	.506	56.7
algorithms/BellmanFord-FunUnsat02.yml | false	false	1.11	.697	54.1	false	.993	.542	65.8
algorithms/BellmanFord-MemSat01.yml | true	TIMEOUT	15.3	14.9	145	unknown	.144	.144	17.1
algorithms/BellmanFord-MemSat02.yml | true	TIMEOUT	15.3	14.9	157	unknown	.161	.161	17.1
algorithms/BellmanFord-MemUnsat01.yml | false	false	1.12	.656	52.5	false	.909	.516	56.8
algorithms/BellmanFord-MemUnsat02.yml | false	false	1.15	.718	53.1	false	.818	.458	56.6
algorithms/BinaryTreeSearch-FunSat01.yml | tru	TIMEOUT	15.3	14.9	578	unknown	.140	.140	17.2
algorithms/BinaryTreeSearch-FunUnsat01.yml |	false	1.00	.566	51.6	false	.945	.514	65.9
algorithms/BinaryTreeSearch-MemSat01.yml | t	TIMEOUT	15.3	14.8	465	unknown	.138	.139	17.1
algorithms/BinaryTreeSearch-MemUnsat01.yml	false	1.00	.559	51.7	unknown	.852	.484	57.1
algorithms/BinaryTreeSearch-MemUnsat02.yml	false	.942	.562	51.9	false	.918	.487	65.9
algorithms/InsertionSort-FunSat01.yml | true	TIMEOUT	15.3	14.8	1920	unknown	.136	.137	17.2
algorithms/InsertionSort-FunSat02.yml | true	TIMEOUT	15.3	14.8	2270	unknown	.133	.134	17.1
algorithms/InsertionSort-FunUnsat01.yml | false	false	7.08	6.69	382	false	.803	.443	56.5
algorithms/InsertionSort-MemSat01.yml | true	TIMEOUT	15.3	14.8	1880	unknown	.138	.139	17.1
algorithms/InsertionSort-MemUnsat01.yml | fals	false	3.30	2.90	455	false	.892	.454	56.7
algorithms/MergeSortIterative-FunSat01.yml | t	TIMEOUT	15.3	14.8	1470	unknown	.149	.150	17.2
algorithms/MergeSortIterative-FunSat02.yml | t	TIMEOUT	15.3	14.8	1710	unknown	.137	.137	17.1
algorithms/MergeSortIterative-FunUnsat01.yml	false	5.34	4.85	1490	unknown	.924	.472	56.9
algorithms/MergeSortIterative-MemSat01.yml |	TIMEOUT	15.3	14.8	1480	unknown	.151	.151	17.1
algorithms/MergeSortIterative-MemUnsat01.yn	TIMEOUT	15.3	14.9	1480	unknown	.135	.135	17.2
algorithms/RedBlackTree-FunSat01.yml | true	TIMEOUT	15.3	14.8	531	unknown	.137	.137	17.2
algorithms/RedBlackTree-FunUnsat01.yml | false	false	1.35	.686	61.2	false	1.07	.517	62.7
algorithms/RedBlackTree-MemSat01.yml | true	TIMEOUT	15.3	14.8	186	unknown	.139	.139	17.2
algorithms/RedBlackTree-MemUnsat01.yml | fal	false	1.23	.699	53.7	false	1.32	.602	80.4
algorithms/SortedListInsert-FunSat01.yml | true	TIMEOUT	15.3	14.8	530	unknown	.137	.137	17.2
algorithms/SortedListInsert-FunSat02.yml | true	TIMEOUT	15.3	14.9	242	unknown	.135	.135	17.2
algorithms/SortedListInsert-FunUnsat01.yml | f	false	2.47	2.07	173	unknown	.824	.443	57.3
algorithms/SortedListInsert-MemSat01.yml | tru	TIMEOUT	15.3	14.9	1410	unknown	.147	.148	17.1
algorithms/SortedListInsert-MemUnsat01.yml |	false	1.02	.572	51.5	unknown	.986	.518	66.1
algorithms/Trie-FunSat01.yml | true	true	2.02	1.57	54.3	unknown	.202	.202	22.9
algorithms/Trie-FunUnsat01.yml | false	false	1.12	.689	52.5	false	.986	.552	57.9
algorithms/Trie-MemSat01.yml | true	true	2.00	1.57	53.1	unknown	.174	.175	22.7
algorithms/Trie-MemUnsat01.yml | false	false	1.02	.590	53.2	false	.920	.517	58.2
algorithms/Tsp-FunSat01.yml | false	TIMEOUT	15.3	14.8	888	unknown	.137	.137	17.2
algorithms/Tsp-FunUnsat01.yml | false	false	2.52	2.11	222	false	.872	.462	57.0
algorithms/Tsp-MemSat01.yml | true	TIMEOUT	15.3	14.8	792	unknown	.137	.137	17.2
algorithms/Tsp-MemUnsat01.yml | false	TIMEOUT	15.3	14.8	798	unknown	.138	.138	17.2
juliet-java/CWE369_Divide_by_Zero__float_co	false	1.11	.634	54.8	false	1.10	.543	77.1
juliet-java/CWE369_Divide_by_Zero__float_co	false	1.16	.662	55.1	false	1.05	.552	61.5
juliet-java/CWE369_Divide_by_Zero__float_co	true	1.82	1.32	56.5	unknown	.148	.148	19.1
juliet-java/CWE369_Divide_by_Zero__float_co	false	1.16	.643	55.3	false	.983	.503	63.2
juliet-java/CWE369_Divide_by_Zero__float_co	false	1.29	.798	55.5	false	.985	.544	61.0
juliet-java/CWE369_Divide_by_Zero__float_co	true	1.90	1.32	56.1	unknown	.158	.158	19.6
juliet-java/CWE369_Divide_by_Zero__float_co	false	1.11	.645	55.5	unknown	1.01	.530	62.0
juliet-java/CWE369_Divide_by_Zero__float_co	false	1.17	.680	55.5	unknown	1.02	.557	62.3
juliet-java/CWE369_Divide_by_Zero__float_co	true	1.80	1.32	55.7	unknown	.165	.165	19.9
jdart-regression/OverapproximationString01.yı	false	1.06	.598	51.5	false	5.43	4.83	2440
jdart-regression/URLDecoder01.yml | true	TIMEOUT	15.3	14.9	828	unknown	.141	.141	17.1
jdart-regression/URLDecoder02.yml | false	TIMEOUT	15.3	14.8	845	unknown	.151	.151	17.3



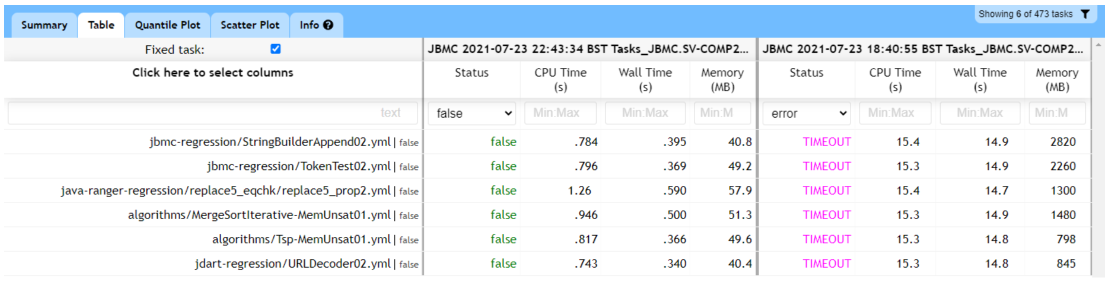
jdart-regression/addition01.yml | false	TIMEOUT	15.3	14.9	405	unknown	.137	.138	17.1
jdart-regression/array-iteration01.yml | false	false	.970	.546	51.9	false	.796	.449	55.6
jdart-regression/boundcheck100.yml | false	false	5.58	5.15	61.5	false	.930	.503	61.6
jdart-regression/boundcheck200.yml | false	false	10.2	9.80	99.9	false	1.03	.557	63.4
jdart-regression/boundcheck30.yml | false	false	3.06	2.54	52.4	false	.938	.484	59.5
jdart-regression/double2long.yml | false	false	.908	.536	51.9	false	.871	.475	55.8
jdart-regression/float.yml | false	false	.915	.528	51.2	false	.820	.450	54.8
jdart-regression/list2.yml | true	true	4.48	4.04	59.3	unknown	.161	.161	24.1
jdart-regression/radians.yml | false	false	1.10	.669	52.9	false	.897	.476	59.8
jdart-regression/shifting.yml | false	false	.933	.575	51.4	false	.946	.479	68.1
jdart-regression/shifting2.yml | false	false	.870	.506	51.3	false	.981	.484	67.4
jdart-regression/shifting3.yml | false	false	.960	.588	51.2	false	.901	.493	58.6
jdart-regression/startswith.yml | true	true	1.48	1.05	51.7	unknown	.144	.144	18.4

Figure 4.6: Full benchmark results

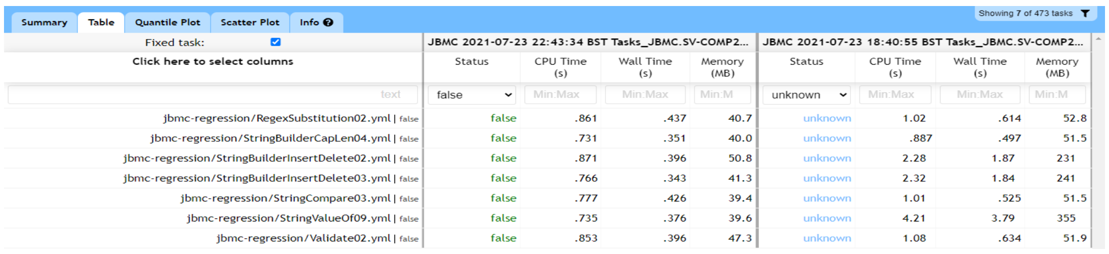
Figure 4.6 shows the complete benchmark result table. The content selected to be displayed is the result after modifying the JBMC script and the result of the witness verification. The left column in the figure lists all the Java benchmarks that have been benchmarked, the middle column shows the JBMC verification results, and the right column shows the witness validation results. In the status cell, the green font represents the correct or expected result, the red font represents the wrong result, the blue font represents the unknown result, and the pink font represents the result of abnormal operation.

As can be seen from the figure, JBMC did not produce an error result, and most of them were correct results while the witness validator produced very few wrong results. In addition, there are two situations for the results shown in blue in the witness validation. One is that the witness file is a correctness-witness, and the other is that the witness contains a counterexample of String object. Both of these witnesses are ignored during validation.

4.4.2 Result analysis



Fixed task: ✓	JBMC 2021-07-23 22:43:34 BST Tasks_JBMC.SV-COMP2...				JBMC 2021-07-23 18:40:55 BST Tasks_JBMC.SV-COMP2...			
Click here to select columns	Status	CPU Time (s)	Wall Time (s)	Memory (MB)	Status	CPU Time (s)	Wall Time (s)	Memory (MB)
text	false	Min:Max	Min:Max	Min:M	error	Min:Max	Min:Max	Min:M
jbmc-regression/StringBuilderAppend02.yml | false	false	.784	.395	40.8	TIMEOUT	15.4	14.9	2820
jbmc-regression/TokenTest02.yml | false	false	.796	.369	49.2	TIMEOUT	15.3	14.9	2260
java-ranger-regression/replace5_eqchk/replace5_prop2.yml | false	false	1.26	.590	57.9	TIMEOUT	15.4	14.7	1300
algorithms/MergeSortIterative-MemUnsat01.yml | false	false	.946	.500	51.3	TIMEOUT	15.3	14.9	1480
algorithms/Tsp-MemUnsat01.yml | false	false	.817	.366	49.6	TIMEOUT	15.3	14.8	798
jdart-regression/URLDecoder02.yml | false	false	.743	.340	40.4	TIMEOUT	15.3	14.8	845



Fixed task: ✓	JBMC 2021-07-23 22:43:34 BST Tasks_JBMC.SV-COMP2...				JBMC 2021-07-23 18:40:55 BST Tasks_JBMC.SV-COMP2...			
Click here to select columns	Status	CPU Time (s)	Wall Time (s)	Memory (MB)	Status	CPU Time (s)	Wall Time (s)	Memory (MB)
text	false	Min:Max	Min:Max	Min:M	unknown	Min:Max	Min:Max	Min:M
jbmc-regression/RegexSubstitution02.yml | false	false	.861	.437	40.7	unknown	1.02	.614	52.8
jbmc-regression/StringBuilderCapLen04.yml | false	false	.731	.351	40.0	unknown	.887	.497	51.5
jbmc-regression/StringBuilderInsertDelete02.yml | false	false	.871	.396	50.8	unknown	2.28	1.87	231
jbmc-regression/StringBuilderInsertDelete03.yml | false	false	.766	.343	41.3	unknown	2.32	1.84	241
jbmc-regression/StringCompare03.yml | false	false	.777	.426	39.4	unknown	1.01	.525	51.5
jbmc-regression/StringValueOf09.yml | false	false	.735	.376	39.6	unknown	4.21	3.79	355
jbmc-regression/Validate02.yml | false	false	.853	.396	47.3	unknown	1.08	.634	51.9

Figure 4.7: Statistics of different JBMC experimental results

This Figure 4.7 shows all the 13 different results of the two JBMC benchmarks. The green word false in the status cell in the left column indicates that the result is expected. In the status cell in the right column, the pink word TIMEOUT indicates that the result is unexpected, and the blue word unknown indicates that the result is unknown.

These 13 Java benchmarks all contain String objects, and as introduced in the background section, JBMC has limited verification capabilities for them. These 13 results indicate that the first correct detection results of the violation assertions corresponding to these Java benchmarks are output by the JVM. Therefore, after ignoring the JVM assertion detection mechanism and directly pass the Java benchmarks to JBMC for verification, JBMC did not find these violation assertions.

Although the modified wrapper script will not be applied to the SV-COMP, the following analysis still focuses on its results and the corresponding witness validation results. Because its witness result set is large, it is more conducive to the analysis of witness validation results for complex programs.

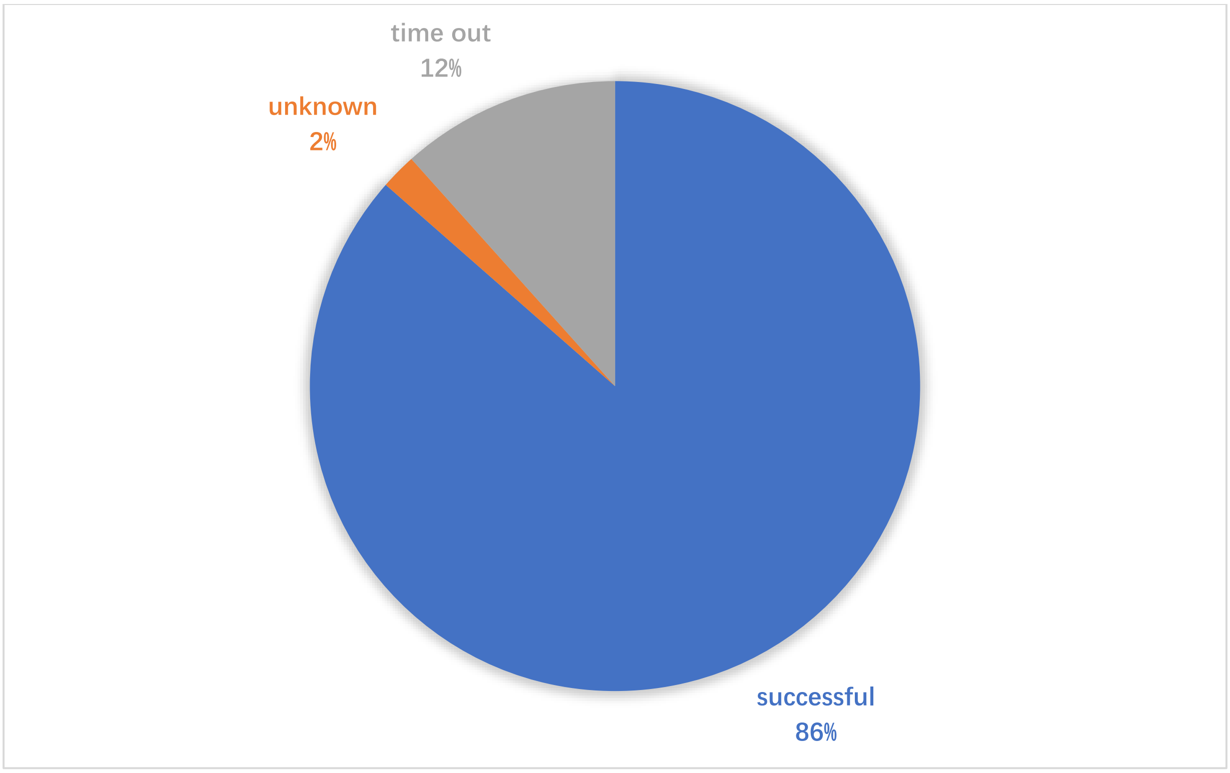


Figure 4.8: Pie chart of results of JBMC

The Figure 4.8 above shows an analysis of the results of the JBMC benchmark. Among these results, unexpected results accounted for 12%, unknown results accounted for 2%, and correct results accounted for 86%.

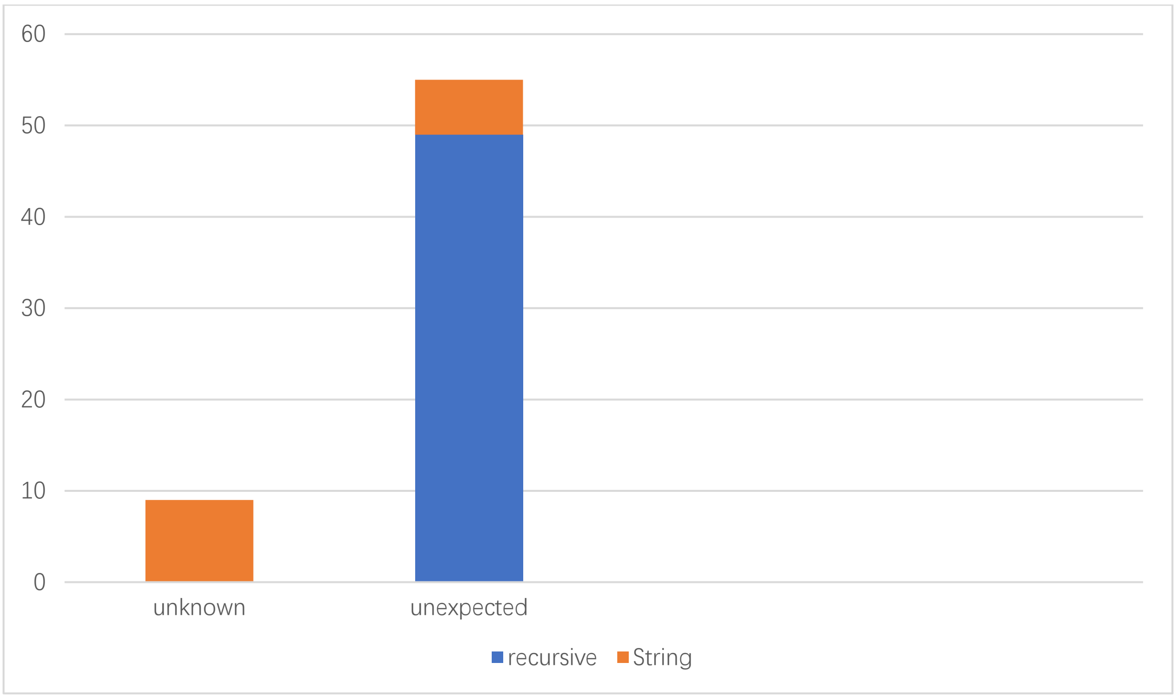


Figure 4.9: Bar graph of non-correct results of JBMC

The Figure 4.9 shows the non-correct (unexpected or unknown) results of the JBMC benchmark. Of a total of 473 Java benchmark results, 64 were non-correct. Among them, the orange bars in the figure indicate that 9 unknown results and 6 unexpected results are caused by the String objects contained in the Java benchmarks, and the blue bars show that 49 Java benchmarks with unexpected results contain recursive functions. Due to the characteristics of bounded model checking, we need to indicate a specific recursion depth when verifying the program, otherwise it will run out of time or memory.

For a Java benchmark containing recursive functions, JBMC cannot prove that it must be a safe program, because it depends on the unwind depth of recursion, which is determined by the user.

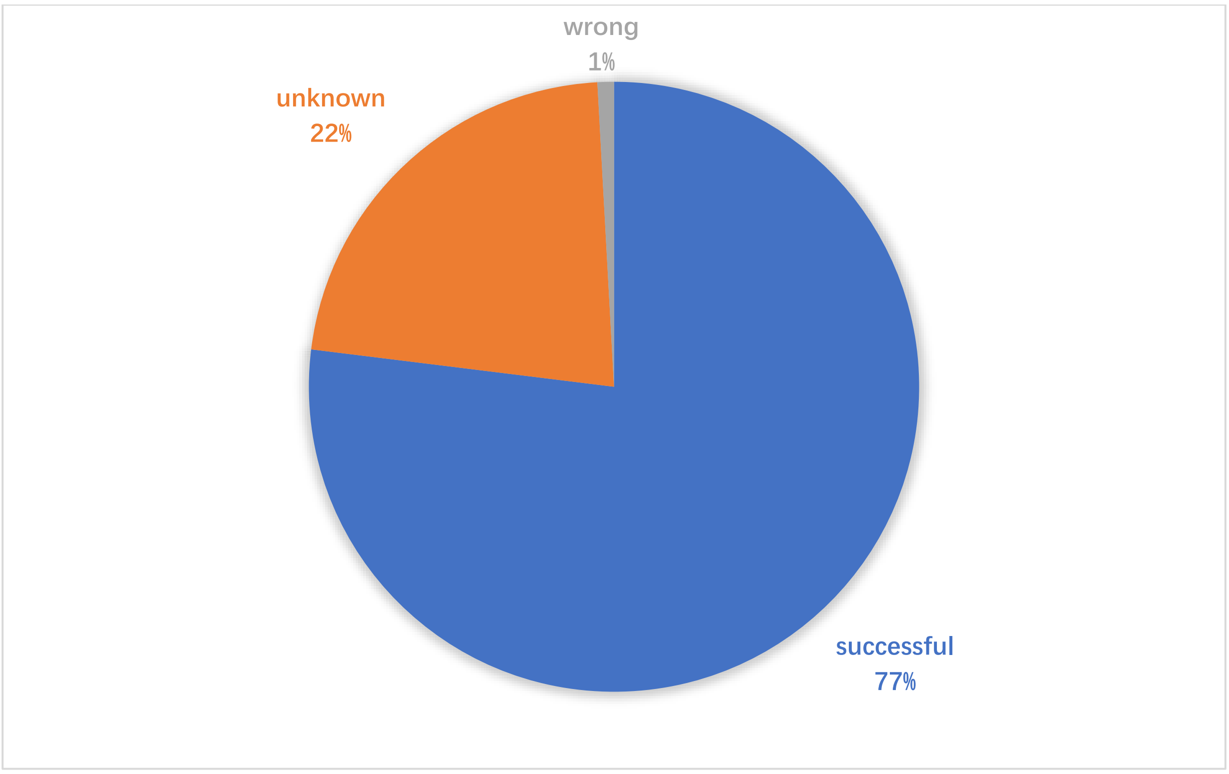
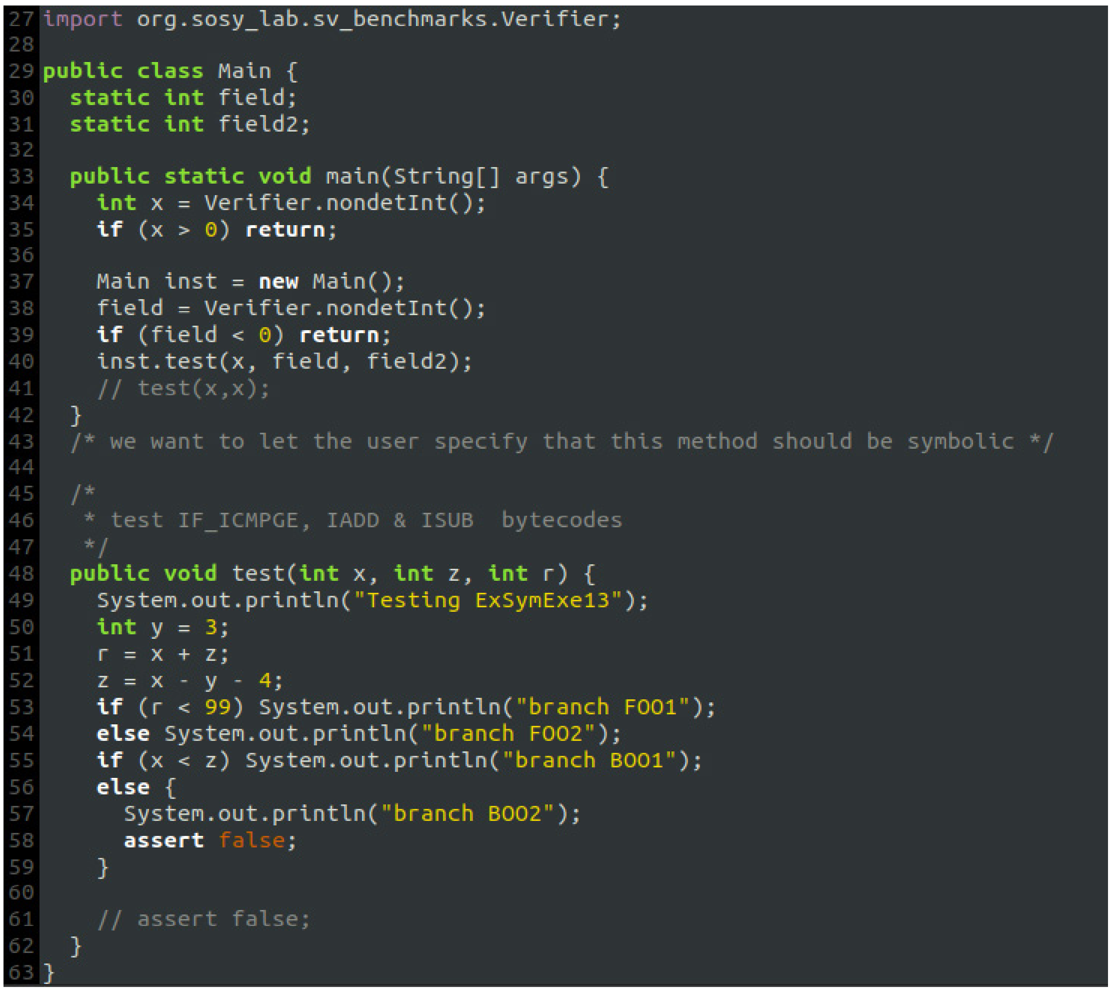


Figure 4.10: Pie chart of witness validation results

The Figure 4.10 shows the result analysis of witness validation. As can be seen from the figure, among the results of witness validation, there are 51 unknown results, accounting for 22% of the total results. The reason for the unknown is that witnesses have no available counterexamples to reproduce the errors detected by JBMC. The

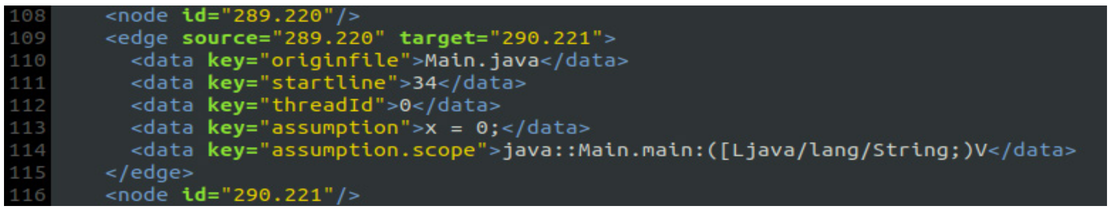
Java benchmarks corresponding to these witnesses are mainly Java programs containing String objects. In addition, there are 2 wrong results, accounting for 1%.

However, when we manually check these wrong results, we will find that these are not caused by witnesses producing false counterexamples. Here is an example to illustrate the case that the validation fails even if the counterexamples are correct.



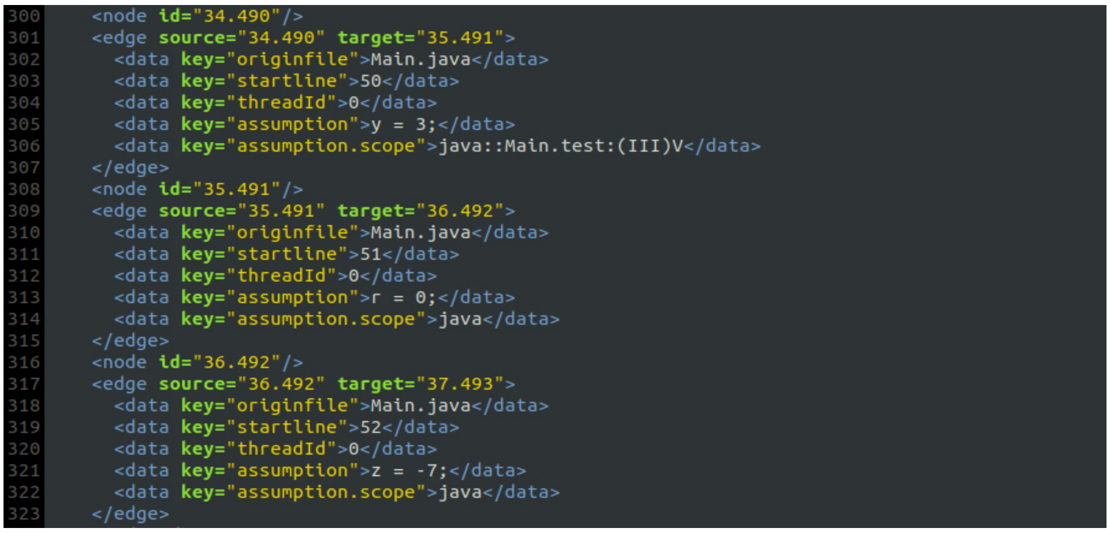
```
27 import org.sosy_lab.sv_benchmarks.Verifier;
28
29 public class Main {
30   static int field;
31   static int field2;
32
33   public static void main(String[] args) {
34     int x = Verifier.nondetInt();
35     if (x > 0) return;
36
37     Main inst = new Main();
38     field = Verifier.nondetInt();
39     if (field < 0) return;
40     inst.test(x, field, field2);
41     // test(x,x);
42   }
43   /* we want to let the user specify that this method should be symbolic */
44
45   /*
46    * test IF_ICMPGE, IADD & ISUB  bytecodes
47    */
48   public void test(int x, int z, int r) {
49     System.out.println("Testing ExSymExe13");
50     int y = 3;
51     r = x + z;
52     z = x - y - 4;
53     if (r < 99) System.out.println("branch FOO1");
54     else System.out.println("branch FOO2");
55     if (x < z) System.out.println("branch BOO1");
56     else {
57       System.out.println("branch BOO2");
58       assert false;
59     }
60
61     // assert false;
62   }
63 }
```

Figure 4.11: An example of complex program



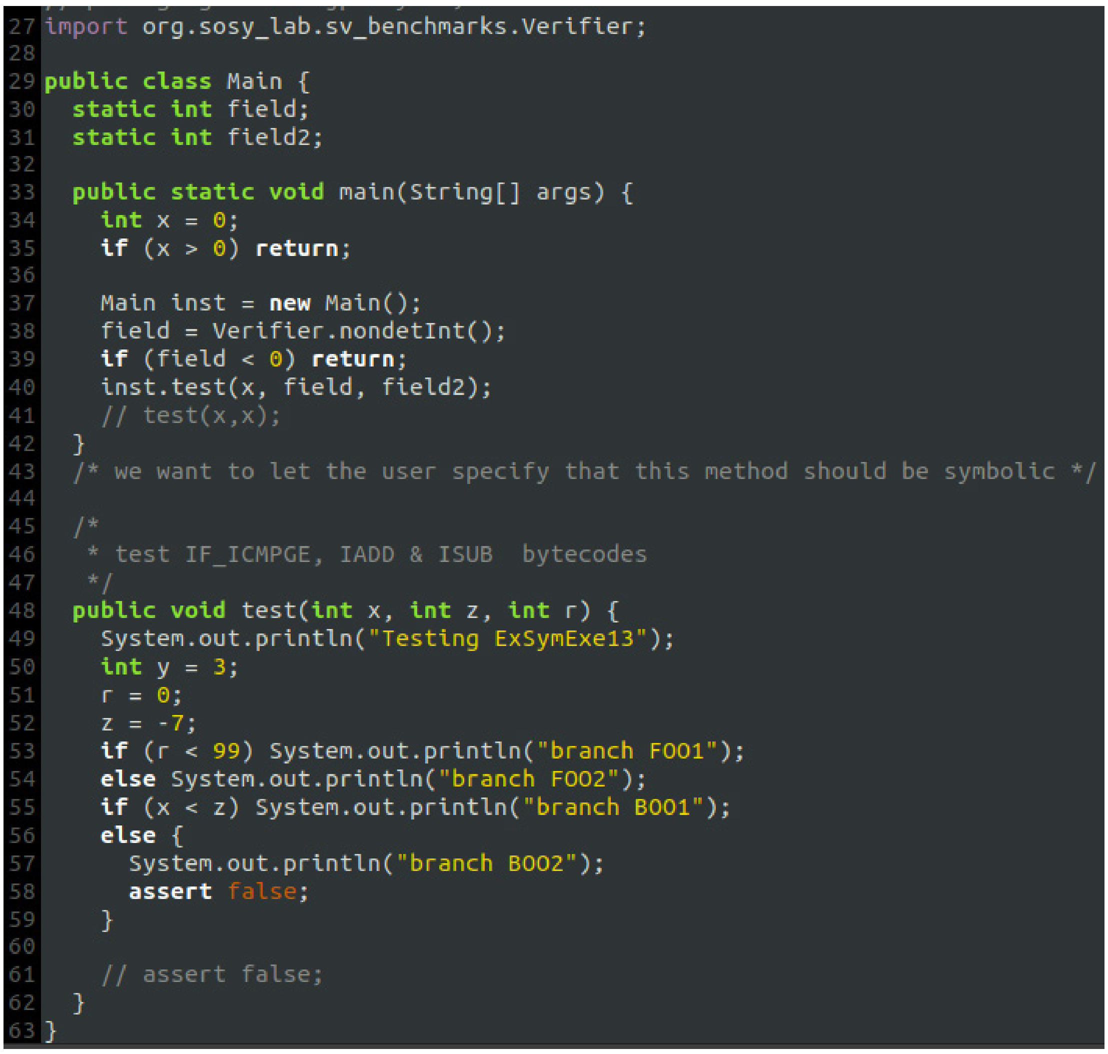
```
    <node id="289.220"/>
    <edge source="289.220" target="290.221">
      <data key="originfile">Main.java</data>
      <data key="startline">34</data>
      <data key="threadId">0</data>
      <data key="assumption">x = 0;</data>
      <data key="assumption.scope">java::Main.main:([Ljava/lang/String;)V</data>
    </edge>
    <node id="290.221"/>
```

Figure 4.12: Witness of the complex program (part 1)



```
  <node id="34.490"/>
  <edge source="34.490" target="35.491">
    <data key="originfile">Main.java</data>
    <data key="startline">50</data>
    <data key="threadId">0</data>
    <data key="assumption">y = 3;</data>
    <data key="assumption.scope">java::Main.test:(III)V</data>
  </edge>
  <node id="35.491"/>
  <edge source="35.491" target="36.492">
    <data key="originfile">Main.java</data>
    <data key="startline">51</data>
    <data key="threadId">0</data>
    <data key="assumption">r = 0;</data>
    <data key="assumption.scope">java</data>
  </edge>
  <node id="36.492"/>
  <edge source="36.492" target="37.493">
    <data key="originfile">Main.java</data>
    <data key="startline">52</data>
    <data key="threadId">0</data>
    <data key="assumption">z = -7;</data>
    <data key="assumption.scope">java</data>
  </edge>
```

Figure 4.13: Witness of the complex program (part 2)



```
27 import org.sosy_lab.sv_benchmarks.Verifier;
28
29 public class Main {
30   static int field;
31   static int field2;
32
33   public static void main(String[] args) {
34     int x = 0;
35     if (x > 0) return;
36
37     Main inst = new Main();
38     field = Verifier.nondetInt();
39     if (field < 0) return;
40     inst.test(x, field, field2);
41     // test(x,x);
42   }
43   /* we want to let the user specify that this method should be symbolic */
44
45   /*
46    * test IF_ICMPGE, IADD & ISUB  bytecodes
47    */
48   public void test(int x, int z, int r) {
49     System.out.println("Testing ExSymExe13");
50     int y = 3;
51     r = 0;
52     z = -7;
53     if (r < 99) System.out.println("branch FOO1");
54     else System.out.println("branch FOO2");
55     if (x < z) System.out.println("branch BOO1");
56     else {
57       System.out.println("branch BOO2");
58       assert false;
59     }
60
61     // assert false;
62   }
63 }
```

Figure 4.14: A new program from the complex program

In this example, the validator successfully uses the counterexample in witness but may produce a wrong result. This program is shown in Figure 4.11. JBMC verified it and found the violation attributes, and generated a violation-witness. Figures 4.12 and 4.13 list the counterexamples in the witness. Figure 4.14 is a new program generated based on the counterexamples and the source program. After checking, we can see that the new program is generated correctly. However, when the new program is compiled and run multiple times to detect assertion errors, the results will be different. This is because there is still a non-deterministic variable `field` in the new program. When its value is less than 0, it will end correctly. Besides, there is no counter-example of the variable `field` in witness, but a counter-example of the variable `z`, which is the parameter variable that the variable `field` is passed to the method `test` as a parameter.

4.5 Threats to validity

No	Benchmark	JBMC	Wit4JBMC	Counterex-ample?	Non-deterministic variables remaining?	Comment
1	jayhorn-recursive/Ackermann01.yml	false	false	Y	N	
2	jayhorn-recursive/InfiniteLoop.yml	false	false	Y	N	
3	jayhorn-recursive/UnsatAckermann01.yml	false	false	Y	N	
4	jayhorn-recursive/UnsatAddition01.yml	false	false	Y	N	
5	jayhorn-recursive/UnsatEvenOdd01.yml	false	false	Y	N	
6	jayhorn-recursive/UnsatFibonacci01.yml	false	false	Y	N	
7	jayhorn-recursive/UnsatFibonacci02.yml	false	false	Y	N	
8	jayhorn-recursive/UnsatMccarthy91.yml	false	false	Y	N	
9	jbmc-regression/ArithmeticException1.yml	false	false	Y	N	
10	jbmc-regression/ArithmeticException6.yml	false	false	Y	N	
11	jbmc-regression/ArrayIndexOutOfBoundsException1.yml	false	false	Y	N	
12	jbmc-regression/ArrayIndexOutOfBoundsException2.yml	false	false	Y	N	
13	jbmc-regression/ArrayIndexOutOfBoundsException3.yml	false	false	N	Y	verifer
14	jbmc-regression/ClassCastException1.yml	false	false	N	N	non
15	jbmc-regression/ClassCastException3.yml	false	false	N	N	non
16	jbmc-regression/NegativeArraySizeException1.yml	false	false	N	N	non
17	jbmc-regression/NegativeArraySizeException2.yml	false	false	N	N	non
18	jbmc-regression/NullPointerException2.yml	false	false	N	N	non
19	jbmc-regression/NullPointerException3.yml	false	false	N	N	non
20	jbmc-regression/NullPointerException4.yml	false	false	N	N	non
21	jbmc-regression/StaticCharMethods04.yml	false	false	Y	N	
22	jbmc-regression/StringContains02.yml	false	false	N	Y	verifer

23	jbmc-regression/StringIndexMethods05.yml	false	false	N	Y	verifer
24	jbmc-regression/StringValueOf04.yml	false	false	Y	N	
25	jbmc-regression/StringValueOf06.yml	false	false	Y	N	
26	jbmc-regression/StringValueOf07.yml	false	false	Y	N	
27	jbmc-regression/assert2.yml	false	false	Y	N	
28	jbmc-regression/assert3.yml	false	false	Y	N	
29	jbmc-regression/assert4.yml	false	false	Y	N	
30	jbmc-regression/athrow1.yml	false	false	N	N	Non
31	jbmc-regression/bug-test-gen-095.yml	false	false	N	Y	verifer
32	jbmc-regression/exceptions1.yml	false	false	N	N	non
33	jbmc-regression/exceptions10.yml	false	false	N	N	non
34	jbmc-regression/exceptions11.yml	false	false	N	N	non
35	jbmc-regression/exceptions12.yml	false	false	N	N	non
36	jbmc-regression/exceptions13.yml	false	false	N	N	non
37	jbmc-regression/exceptions16.yml	false	false	N	N	non
38	jbmc-regression/exceptions2.yml	false	false	N	N	non
39	jbmc-regression/exceptions3.yml	false	false	N	N	non
40	jbmc-regression/exceptions6.yml	false	false	N	N	non
41	jbmc-regression/exceptions7.yml	false	false	N	N	non
42	jbmc-regression/exceptions8.yml	false	false	N	N	non
43	jbmc-regression/interface1.yml	false	false	N	N	non
44	jbmc-regression/java_append_char.yml	false	false	Y	N	
45	jbmc-regression/return1.yml	false	false	Y	N	
46	jbmc-regression/return2.yml	false	false	Y	N	
47	jbmc-regression/virtual2.yml	false	false	N	N	non
48	jpf-regression/ExDarko_false.yml	false	false	N	Y	verifer
49	jpf-regression/ExException_false.yml	false	false	N	Y	verifer
50	jpf-regression/ExGenSymExe_false.yml	false	false	N	Y	verifer
51	jpf-regression/ExLazy_false.yml	false	false	N	Y	verifer
52	jpf-regression/ExSymExe10_false.yml	false	false	Y	Y	verifer
53	jpf-regression/ExSymExe11_false.yml	false	false	Y	Y	verifer
54	jpf-regression/ExSymExe12_false.yml	false	false	Y	Y	verifer
55	jpf-regression/ExSymExe13_false.yml	false	false	Y	Y	verifer
56	jpf-regression/ExSymExe15_false.yml	false	false	Y	Y	verifer
57	jpf-regression/ExSymExe16_false.yml	false	false	Y	N	
58	jpf-regression/ExSymExe17_false.yml	false	false	Y	N	
59	jpf-regression/ExSymExe18_false.yml	false	false	Y	N	
60	jpf-regression/ExSymExe19_false.yml	false	false	Y	Y	verifer
61	jpf-regression/ExSymExe1_false.yml	false	true	Y	Y	verifer
62	jpf-regression/ExSymExe20_false.yml	false	false	Y	Y	verifer
63	jpf-regression/ExSymExe21_false.yml	false	false	Y	Y	verifer
64	jpf-regression/ExSymExe25_false.yml	false	false	Y	N	
65	jpf-regression/ExSymExe26_false.yml	false	false	Y	N	
66	jpf-regression/ExSymExe27_false.yml	false	false	Y	N	
67	jpf-regression/ExSymExe28_false.yml	false	false	Y	N	
68	jpf-regression/ExSymExe29_false.yml	false	false	Y	Y	verifer
69	jpf-regression/ExSymExe2_false.yml	false	false	Y	N	

70	jpf-regression/ExSymExe3_false.yml	false	false	Y	N	
71	jpf-regression/ExSymExe4_false.yml	false	false	Y	N	
72	jpf-regression/ExSymExe5_false.yml	false	false	Y	N	
73	jpf-regression/ExSymExe6_false.yml	false	false	Y	Y	verifer
74	jpf-regression/ExSymExe7_false.yml	false	false	Y	N	
75	jpf-regression/ExSymExe8_false.yml	false	false	Y	Y	verifer
76	jpf-regression/ExSymExe9_false.yml	false	false	Y	N	
77	jpf-regression/ExSymExeArrays_false.yml	false	false	Y	N	
78	jpf-regression/ExSymExeBool_false.yml	false	false	Y	N	
79	jpf-regression/ExSymExeComplexMath_false.yml	false	false	Y	N	
80	jpf-regression/ExSymExeD2I_false.yml	false	false	Y	N	
81	jpf-regression/ExSymExeD2L_false.yml	false	false	Y	N	
82	jpf-regression/ExSymExeF2I_false.yml	false	false	Y	N	
83	jpf-regression/ExSymExeF2L_false.yml	false	false	Y	N	
84	jpf-regression/ExSymExeFNEG_false.yml	false	false	Y	N	
85	jpf-regression/ExSymExeGetStatic_false.yml	false	false	N	N	non
86	jpf-regression/ExSymExeI2D_false.yml	false	false	Y	N	
87	jpf-regression/ExSymExeI2F_false.yml	false	false	N	N	non
88	jpf-regression/ExSymExeLCMP_false.yml	false	false	Y	N	
89	jpf-regression/ExSymExeLongBytecodes_false.yml	false	false	Y	N	
90	jpf-regression/ExSymExeResearch_false.yml	false	false	Y	N	
91	jpf-regression/ExSymExeSimple_false.yml	false	false	Y	N	
92	jpf-regression/ExSymExeSuzette_false.yml	false	false	Y	Y	verifer
93	jpf-regression/ExSymExeSwitch_false.yml	false	false	Y	N	
94	jpf-regression/ExSymExeTestAssignments_false.yml	false	false	Y	N	
95	jpf-regression/ExSymExeTestClassFields_false.yml	false	true	N	Y	verifer
96	jpf-regression/ExSymExe_false.yml	false	false	Y	N	
97	jpf-regression/TestLazy_false.yml	false	false	N	Y	verifer
98	java-ranger-regression/WBS/WBS_prop1.yml	false	false	Y	N	
99	java-ranger-regression/WBS/WBS_prop3.yml	false	false	Y	N	
100	java-ranger-regression/WBS/WBS_prop4.yml	false	false	Y	N	
101	MinePump/spec1-5_product1.yml	false	false	Y	N	
102	MinePump/spec1-5_product11.yml	false	false	Y	N	
103	MinePump/spec1-5_product12.yml	false	false	Y	N	
104	MinePump/spec1-5_product14.yml	false	false	Y	N	
105	MinePump/spec1-5_product15.yml	false	false	Y	N	
106	MinePump/spec1-5_product17.yml	false	false	Y	N	
107	MinePump/spec1-5_product18.yml	false	false	Y	N	
108	MinePump/spec1-5_product19.yml	false	false	Y	N	
109	MinePump/spec1-5_product2.yml	false	false	Y	N	
110	MinePump/spec1-5_product20.yml	false	false	Y	N	
111	MinePump/spec1-5_product21.yml	false	false	Y	N	
112	MinePump/spec1-5_product22.yml	false	false	Y	N	
113	MinePump/spec1-5_product23.yml	false	false	Y	N	
114	MinePump/spec1-5_product24.yml	false	false	Y	N	
115	MinePump/spec1-5_product25.yml	false	false	Y	N	
116	MinePump/spec1-5_product26.yml	false	false	Y	N	

117	MinePump/spec1-5_product27.yml	false	false	Y	N	
118	MinePump/spec1-5_product28.yml	false	false	Y	N	
119	MinePump/spec1-5_product29.yml	false	false	Y	N	
120	MinePump/spec1-5_product3.yml	false	false	Y	N	
121	MinePump/spec1-5_product30.yml	false	false	Y	N	
122	MinePump/spec1-5_product31.yml	false	false	Y	N	
123	MinePump/spec1-5_product32.yml	false	false	Y	N	
124	MinePump/spec1-5_product33.yml	false	false	Y	N	
125	MinePump/spec1-5_product34.yml	false	false	Y	N	
126	MinePump/spec1-5_product35.yml	false	false	Y	N	
127	MinePump/spec1-5_product36.yml	false	false	Y	N	
128	MinePump/spec1-5_product37.yml	false	false	Y	N	
129	MinePump/spec1-5_product38.yml	false	false	Y	N	
130	MinePump/spec1-5_product39.yml	false	false	Y	N	
131	MinePump/spec1-5_product4.yml	false	false	Y	N	
132	MinePump/spec1-5_product40.yml	false	false	Y	N	
133	MinePump/spec1-5_product42.yml	false	false	Y	N	
134	MinePump/spec1-5_product43.yml	false	false	Y	N	
135	MinePump/spec1-5_product44.yml	false	false	Y	N	
136	MinePump/spec1-5_product45.yml	false	false	Y	N	
137	MinePump/spec1-5_product46.yml	false	false	Y	N	
138	MinePump/spec1-5_product47.yml	false	false	Y	N	
139	MinePump/spec1-5_product48.yml	false	false	Y	N	
140	MinePump/spec1-5_product49.yml	false	false	Y	N	
141	MinePump/spec1-5_product5.yml	false	false	Y	N	
142	MinePump/spec1-5_product50.yml	false	false	Y	N	
143	MinePump/spec1-5_product51.yml	false	false	Y	N	
144	MinePump/spec1-5_product52.yml	false	false	Y	N	
145	MinePump/spec1-5_product53.yml	false	false	Y	N	
146	MinePump/spec1-5_product54.yml	false	false	Y	N	
147	MinePump/spec1-5_product55.yml	false	false	Y	N	
148	MinePump/spec1-5_product56.yml	false	false	Y	N	
149	MinePump/spec1-5_product6.yml	false	false	Y	N	
150	MinePump/spec1-5_product7.yml	false	false	Y	N	
151	MinePump/spec1-5_product8.yml	false	false	Y	N	
152	MinePump/spec1-5_product9.yml	false	false	Y	N	
153	algorithms/BellmanFord-FunUnsat02.yml	false	false	Y	Y	verifer
154	algorithms/BellmanFord-MemUnsat01.yml	false	false	Y	Y	verifer
155	algorithms/BellmanFord-MemUnsat02.yml	false	false	Y	N	
156	algorithms/BinaryTreeSearch-FunUnsat01.yml	false	false	Y	N	
157	algorithms/BinaryTreeSearch-MemUnsat02.yml	false	false	Y	Y	verifer
158	algorithms/InsertionSort-FunUnsat01.yml	false	false	Y	Y	verifer
159	algorithms/InsertionSort-MemUnsat01.yml	false	false	Y	N	
160	algorithms/RedBlackTree-FunUnsat01.yml	false	false	Y	N	
161	algorithms/RedBlackTree-MemUnsat01.yml	false	false	Y	N	
162	algorithms/Trie-FunUnsat01.yml	false	false	Y	N	
163	algorithms/Trie-MemUnsat01.yml	false	false	N	Y	verifer

164	algorithms/Tsp-FunUnsat01.yml	false	false	Y	N	
165	juliet-ja-va/CWE369_Divide_by_Zero__float_connect_tcp_divide_01_bad.yml	false	false	N	Y	verifer
166	juliet-ja-va/CWE369_Divide_by_Zero__float_connect_tcp_divide_01_bad_version2.yml	false	false	N	Y	verifer
167	juliet-ja-va/CWE369_Divide_by_Zero__float_connect_tcp_divide_81a_bad.yml	false	false	N	Y	verifer
168	juliet-ja-va/CWE369_Divide_by_Zero__float_connect_tcp_divide_81a_bad_version2.yml	false	false	N	Y	verifer
169	jdart-regression/OverapproximationString01.yml	false	false	N	Y	verifer
170	jdart-regression/array-iteration01.yml	false	false	Y	Y	verifer
171	jdart-regression/boundcheck100.yml	false	false	Y	N	
172	jdart-regression/boundcheck200.yml	false	false	Y	N	
173	jdart-regression/boundcheck30.yml	false	false	Y	N	
174	jdart-regression/double2long.yml	false	false	Y	N	
175	jdart-regression/float.yml	false	false	Y	N	
176	jdart-regression/radians.yml	false	false	Y	N	
177	jdart-regression/shifting.yml	false	false	Y	N	
178	jdart-regression/shifting2.yml	false	false	Y	N	
179	jdart-regression/shifting3.yml	false	false	Y	N	

Table 4.1: Statistics of witness validation results

Table 4.1 shows the non-unknown (true or false) results of witness validation against the benchmarks. There are six columns in the table: “Benchmark”, “JBMC”, “Wit4JBMC”, “Counterexample?”, “Non-deterministic variables remaining?” and “Comment”. The “Benchmark” column indicates the task-definition name of the Java program. The “JBMC” column indicates the verification result of JBMC, all of which are supposed to be “false”. The “witness validation” column indicates the witness validation result. The “counterexample” column indicates if there are counterexamples in the violation-witness file using “Y” for yes and “N” for no. The “Non-deterministic variable remaining?” indicates if there are still non-deterministic variables in the new generated Java programs using “Y” for yes and “N” for no.

The “Comment” section provides more detail about the result. If the column is empty, it means that the witness validator correctly used the counterexamples in the witness and reproduced the assertion error according to the newly generated program. If the content of the comment is “verifier”, it means that there are still non-deterministic values in the newly generated program. As a result, although some programs are not affected by this non-deterministic value, they can still reproduce the assertion error, for other programs, they will only randomly arrive at the statement that asserts the error. Although there are not enough counterexamples in witnesses for these programs, no witnesses have been found to provide wrong counterexamples. Just like the previous example, even though the witness validation failed, it was actually due to insufficient counterexamples, not counterexample errors. If the column is “non”, it means that there is no basic variable type with non-deterministic value in the source program, so there is no counterexample in witness. The content of the newly generated program is the same as that of the source program. In fact, the detected assertion error can be reproduced by directly running the source program.

4.6 Conclusion

The project implemented an appropriate extension to use JBMC to automatically verify the security of all the Java benchmarks in the SV-COMP and then validate their results through witnesses to increase the trustworthiness of JBMC. The most important achievement is the newly designed witness verification tool for Java. From the results of the previous subchapter, it can be seen that this witness validator can correctly validate witnesses corresponding to the most of the Java benchmarks in the SV-COMP. At the same time, although the violation-witness generated by JBMC may be inadequate for some complex Java benchmarks, we have not found a false counterexample in the violation-witness, which means that JBMC did not generate a false alarm.

Soundness and completeness of JBMC. JBMC is complete since no false alarms were found. And it is not sound enough because it failed to detect vulnerabilities in some Java benchmarks, especially for those programs that contain recursions and loops.

5. Further Work

This witness validator still has room for improvement. The first is the validation of the correctness-witness, which contains the path that guides the validator to the safe property. The trustworthiness of JBMC can be further improved by validating the correctness-witness. Secondly, there are deficiencies in the validation of violation-witness. When the value of the non-deterministic variable in the program is not directly given in the witness, the current algorithm cannot ensure the correct validation result. However, the variable values of other counterexamples in witness may be calculated mathematically for the non-deterministic variable. Therefore, if you insist on using this validation algorithm, you can calculate the value of the non-deterministic variable according to the relevant mathematical formula, so as to continue to improve the success rate of validation. Thirdly, strictly speaking, this witness validation algorithm does not read and use all the edges in the violation-witness to detect the violation attributes in the program step by step. It just obtains the edge nodes with counterexamples to generate a new program, and finally uses the JVM to detect the corresponding violation assertions in the program. Therefore, a more rigorous validation algorithm should be designed such that it can check the statement in the program corresponding to the current edge along each edge of the witness, and finally find the violation attribute or no violation attribute, which also achieves both the violation-witness and the correctness-witness validation. Finally, as mentioned in the background chapter, NitWit has the best validation performance in the C language. It interprets the current statement immediately without waiting for the compilation of the complete program, which provides a new idea for optimizing witness validation algorithm for Java, that is, it does not generate and run a new Java program, but interprets the current the corresponding source program statement in the witness.

Except for the improvements mentioned above, this algorithm does not currently support programs containing Java String objects. This is because of JBMC has restrictions on String manipulation. Therefore, it is a remarkable achievement to study JBMC in depth to solve this limitation.

This witness validation tool can be used as the first Java witness validator to contribute to the SV-COMP. However, the technology of generating witnesses by Java verifier has not been widely used. At present, among all the Java verifiers in the SV-COMP, only JBMC provides the function of generating witnesses. Therefore, it cannot be proved whether this tool can validate the witnesses generated by verifiers other than JBMC.

Bibliography

[1] PYPL. "PYPL PopularitY of Programming Language." https://pypl.github.io/PYPL.html (accessed 30 April, 2021).

[2] Computerworld. "Top software failures in recent history." https://www.computerworld.com/article/3412197/top-software-failures-in-recent-history.html#slide2 (accessed 27 April, 2021).

[3] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu, "Bounded model checking," 2003.

[4] L. Cordeiro, P. Kesseli, D. Kroening, P. Schrammel, and M. Trtik, "JBMC: A Bounded Model Checking Tool for Verifying Java Bytecode," Cham, 2018: Springer International Publishing, in Computer Aided Verification, pp. 183-190.

[5] D. Beyer, M. Dangl, D. Dietsch, M. Heizmann, and A. Stahlbauer, "Witness validation and stepwise testification across software verifiers," presented at the Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, Bergamo, Italy, 2015. [Online]. Available: https://doi.org/10.1145/2786805.2786867.

[6] V. L. Tan, "SECURITY ANALYSER TOOL FOR FINDING VULNERABILITIES IN JAVA PROGRAMS," THE UNIVERSITY OF MANCHESTER, 2020. [Online]. Available: https://ssvlab.github.io/lucasccordeiro/supervisions/msc_thesis_vi.pdf

[7] D. Beyer. "BenchExec: A Framework for Reliable Benchmarking and Resource Measurement." https://github.com/sosy-lab/benchexec (accessed 30 April, 2021).

[8] Statista. "Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025." https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/ (accessed 30 April, 2021).

[9] P. Mallik and O. P. Jena, "Analysis of Security Vulnerabilities of Internet of Things and It's Solutions," Singapore, 2021: Springer Singapore, in Intelligent Systems, pp. 393-402.

[10] T. D. Diwan, "An EXPERIMENTAL ANALYSIS OF SECURITY VULNERABILITIES IN INDUSTRIAL INTERNET OF THINGS SERVICES," *INFORMATION TECHNOLOGY IN INDUSTRY,* vol. 9, no. 3, pp. 592-598, 2021.

[11] G. Blaine, in *SonicWall: Encrypted Attacks, IoT Malware Surge as Global Malware Volume Dips*, ed. SonicWall, 2019.

[12] portswigger. "SQL injection." https://portswigger.net/web-security/sql-injection#retrieving-hidden-data (accessed 15 March, 2021).

[13] E. Katz. "Top 10 Most Common Java Vulnerabilities You Need to Prevent." https://spectralops.io/blog/top-10-most-common-java-vulnerabilities-you-need-to-prevent/ (accessed 14 April, 2021).

[14] G. O'Regan, *Concise guide to software testing*. Springer (in eng), 2019.

[15] B. Beizer, *Software testing techniques*, 2nd ed. ed. Van Nostrand Reinhold (in eng), 1990.

[16] S. Nidhra and J. Dondeti, "Black box and white box testing techniques-a literature review," *International Journal of Embedded Systems and Applications (IJESA),* vol. 2, no. 2, pp. 29-50, 2012.

[17] L. Luo, "Software Testing Techniques," *Institute for software research international Carnegie mellon university Pittsburgh, PA,* vol. 15232, p. 19, 2021.

[18] TACAS. "Results of the Competition." https://sv-comp.sosy-lab.org/2021/results/results-verified/ (accessed August 25, 2021).

[19] L. Cordeiro, D. Kroening, and P. Schrammel, "JBMC: Bounded Model Checking for Java Bytecode," Cham, 2019: Springer International Publishing, in Tools and Algorithms for the Construction and Analysis of Systems, pp. 219-223.

[20] S. Anand, C. S. Păsăreanu, and W. Visser, "JPF–SE: A Symbolic Execution Extension to Java PathFinder," Berlin, Heidelberg, 2007: Springer Berlin Heidelberg, in Tools and Algorithms for the Construction and Analysis of Systems, pp. 134-138.

[21] T. Kahsai, P. Rümmer, H. Sanchez, and M. Schäf, "JayHorn: A Framework for Verifying Java programs," Cham, 2016: Springer International Publishing, in Computer Aided Verification, pp. 352-358.

[22] D. Beyer. "Exchange Format for Violation Witnesses and Correctness Witnesses." https://github.com/sosy-lab/sv-witnesses (accessed 30 May, 2021).

[23] D. Beyer and M. Spiessl, "MetaVal: Witness Validation via Verification," Cham, 2020: Springer International Publishing, in Computer Aided Verification, pp. 165-177.

[24] J. Švejda, P. Berger, and J.-P. Katoen, "Interpretation-based violation witness validation for C: NitWit," *Tools and Algorithms for the Construction and Analysis of Systems,* vol. 12078, p. 40, 2020.

[25] Oracle. "Java Platform, Standard Edition Java Shell User's Guide." https://docs.oracle.com/javase/9/jshell/introduction-jshell.htm#JSHEL-GUID-630F27C8-1195-4989-9F6B-2C51D46F52C8 (accessed 10 May, 2021).

[26] Mockito. "Tasty mocking framework for unit tests in Java." https://site.mockito.org/ (accessed 2 May, 2021).

[27] baeldung. "Introduction to PowerMock." https://www.baeldung.com/intro-to-powermock (accessed 2 May, 2021).

[28] D. Beyer. "BenchExec: benchexec." https://github.com/sosy-lab/benchexec/blob/master/doc/benchexec.md (accessed 28 July, 2021).

[29] D. Beyer. "BenchExec: Tool Integration." https://github.com/sosy-lab/benchexec/blob/master/doc/tool-integration.md (accessed 30 July, 2021).