JBMC: Bounded Model Checking for Java Bytecode

Lucas C. Cordeiro[1], **Daniel Kroening**[2, 4], **Peter Schrammel**[3, 4]

[1]University of Manchester, [2]University of Oxford, [3]University of Sussex, [4]Diffblue Ltd

1 Motivation

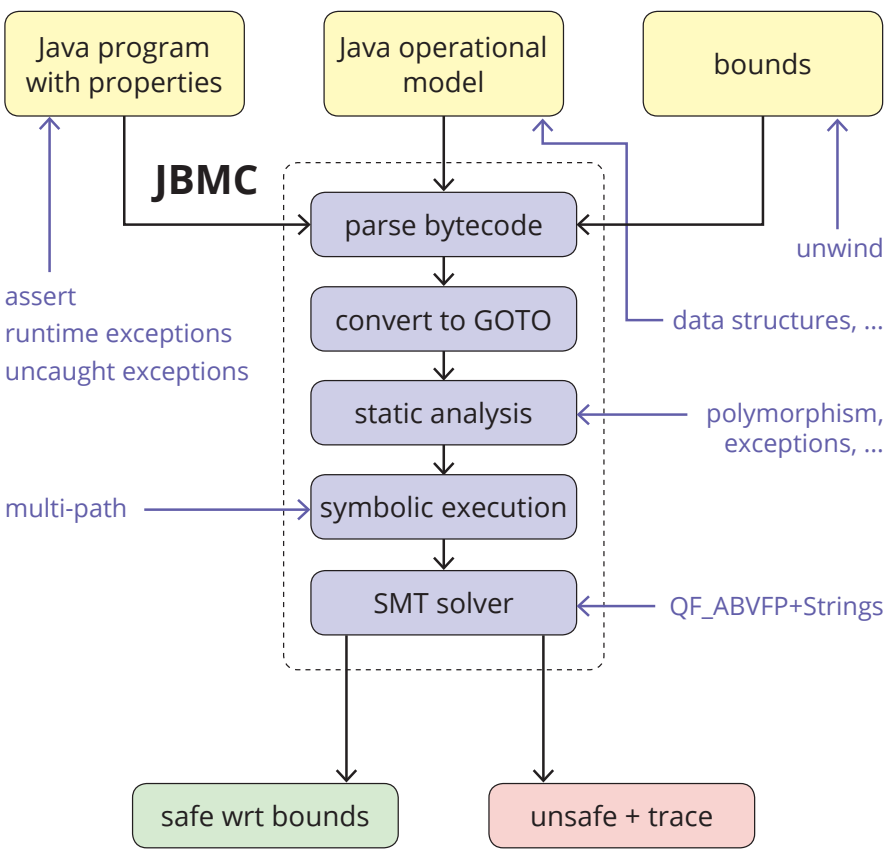
The Java Programming Language is a general-purpose, concurrent, strongly typed, object-oriented language. **Java programs** may have bugs, which may result in array bound violations, unintended arithmetic overflows, and other kinds of functional and runtime errors. Also, Java allows multi-threading, and thus, problems such as race conditions and deadlocks can occur.

2 Approach and Uniqueness

JBMC is an extension to the C Bounded Model Checker (CBMC) [3], named JBMC, to verify Java bytecode [1,2].

JBMC consists of a frontend for parsing Java bytecode and a **Java operational model** (JOM), an exact, but verification-friendly model of the standard Java libraries.

A distinct feature of JBMC, when compared with other approaches, is the use of **Bounded Model Checking** (BMC) in combination with Boolean Satisfiability Modulo Theories (SMT); it symbolically explores the state-space to perform a bit-accurate verification of Java programs.



3 Features

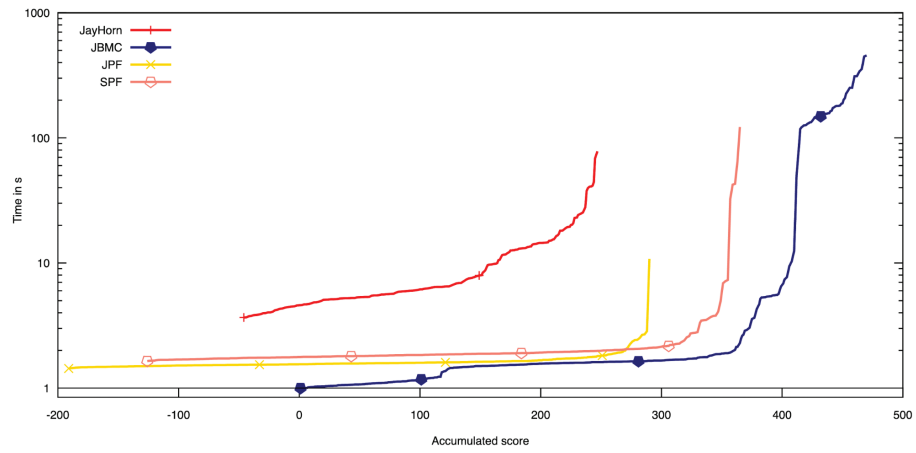
The Java operational model (JOM) consists of simplified models of the most common classes from java.lang and a few from java.util.

JBMC also implements a **solver for strings** to determine the satisfiability of a set of constraints involving strings (based on [5]).

JBMC provides API classes that allow users to define **non-deterministic verification harnesses** and **stub functions** as used in the SV-COMP benchmarks.

4 Strengths and Weaknesses

JBMC does **not produce any incorrect result** for any of the Java verification tasks available in SV-COMP 2019 [4]; it correctly claims **139 benchmarks correct** and **finds bugs in 192**.



However, JBMC aborts (or returns unknown) on 37 benchmarks due to time or memory exhaustion, or due to missing models of the Java standard library.

JBMC's concurrency support is still limited and there is no support for lambdas, reflection and the Java Native Interface (JNI).

5 Tool Setup

The competition submission is based on JBMC version 5.10. For the competition, JBMC is called from a wrapper script.

The **wrapper script** compiles the .java source files in the given benchmark directories and then invokes the JBMC binary repeatedly with **increasing values for the unwinding bound** until the property has been refuted (answering false) or the program has been fully unwound without finding a property violation (answering true).

```
1 import org.sosy_lab.sv_benchmarks.Verifier;
2 public class Main {
3   public static void main (String [] args) {
4     String arg = Verifier.nondetString();
5     float floatValue = Float.parseFloat(arg)
6     String tmp = String.valueOf (floatValue);
7     assert temp.equals("2.50");
8   }
9 }
```

We can run the JBMC wrapper script to check for a reachability property in the program shown above by executing the following command:

```
.\jbmc --propertyfile <path-to-sv-benchmarks>/properties/
assert.prp <path-to-sv-benchmarks>/java/jbmc-regression/
StringValueOf08
```

6 Software Project

JBMC is maintained by Peter Schrammel together with numerous contributors from the community.

It is publicly available under a BSD-style license.

The source code is available at **http://www.github.com/diffblue/cbmc** in the jbmc directory.

Instructions for building JBMC from source are given in the file COMPILING.md.

References:

[1] Cordeiro, L.C., Kroening, D., Schrammel, P. JBMC: A bounded model checking tool for verifying Java bytecode (Competition Contribution). In: TACAS. LNCS, vol. 11429. Springer (2019)

[2] Cordeiro, L.C., Kesseli, P., Kroening, D., Schrammel, P., Trtík, M.: JBMC: A bounded model checking tool for verifying Java bytecode. In: CAV. LNCS, vol. 10981, pp. 183–190. Springer (2018)

[3] Clarke, E.M., Kroening, D., Lerda, F.: A tool for checking ANSI-C programs. In: Tools and Algorithms for the Construction and Analysis of Systems, TACAS. LNCS, vol. 2988, pp. 168–176. Springer (2004)

[4] Beyer, D.: Automatic Verification of C and Java Programs: SV-COMP 2019, TACAS. LNCS, vol. 11429. Springer (2019)

[5] Li, G., Ghosh, I.: PASS: string solving with parameterized array and interval automaton. In: HVC. LNCS, vol. 8244, pp. 15–31 (2013)