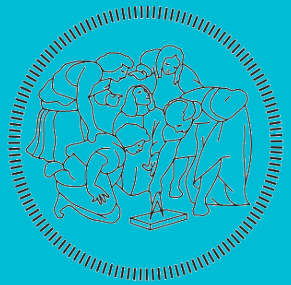


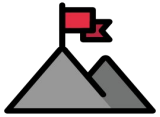
POLITECNICO
MILANO 1863

JBMC

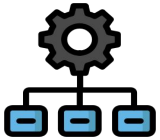
Java Bounded Model Checker

lnghrdntcr@github
nicofossa@github

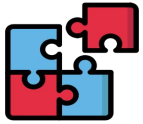
OVERVIEW



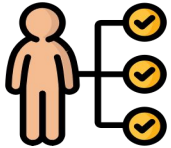
1. Introduction



2. Architecture

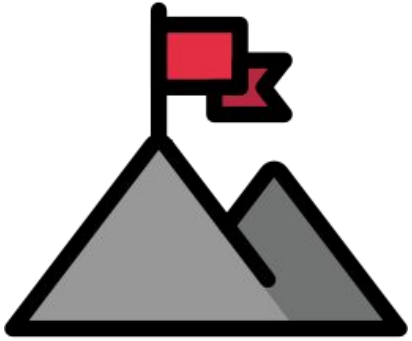


3. Programming by contract



4. Examples

5. Pros / Cons



1.

Introduction

What is Bounded Model Checking?

- Leverage the strengths of SAT solvers
- Since counterexamples to LTL properties have finite length find a path of finite length up to k that violates the property!
- "Φ_k is satisfiable ⇔ there is a counterexample to the property P on TS of length at most k"

What is JBMC?

- Bounded model checking tool
- Java programs verifier
- Frontend of CBMC

What it can be used for? (generally)

- Find property violation (up to a given bound k)
- Verify user defined assertions
  - Auto generates assertions for common pitfalls

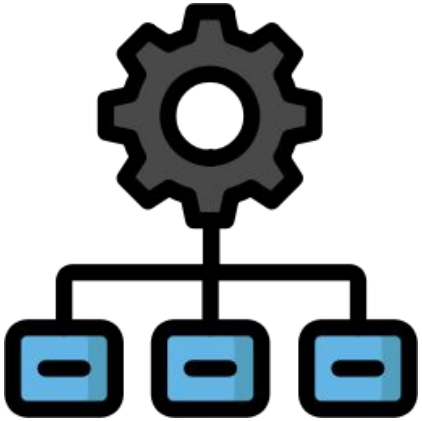
What it can be used for? (specifically)

- Array bounds violations
- Unintended arithmetic overflows
- Functional / Runtime Errors
- …

```
java.lang.ArrayIndexOutOfBoundsException: 3 >= 3
    at java.util.Vector.elementAt(Vector.java:427)
    at junit.samples.VectorTest.testElementAt(Vector
    at sun.reflect.NativeMethodAccessorImpl.invoke0(
    at sun.reflect.NativeMethodAccessorImpl.invoke(N
    at sun.reflect.DelegatingMethodAccessorImpl.invo
    at java.lang.reflect.Method.invoke(Method.java:3
```

On what?

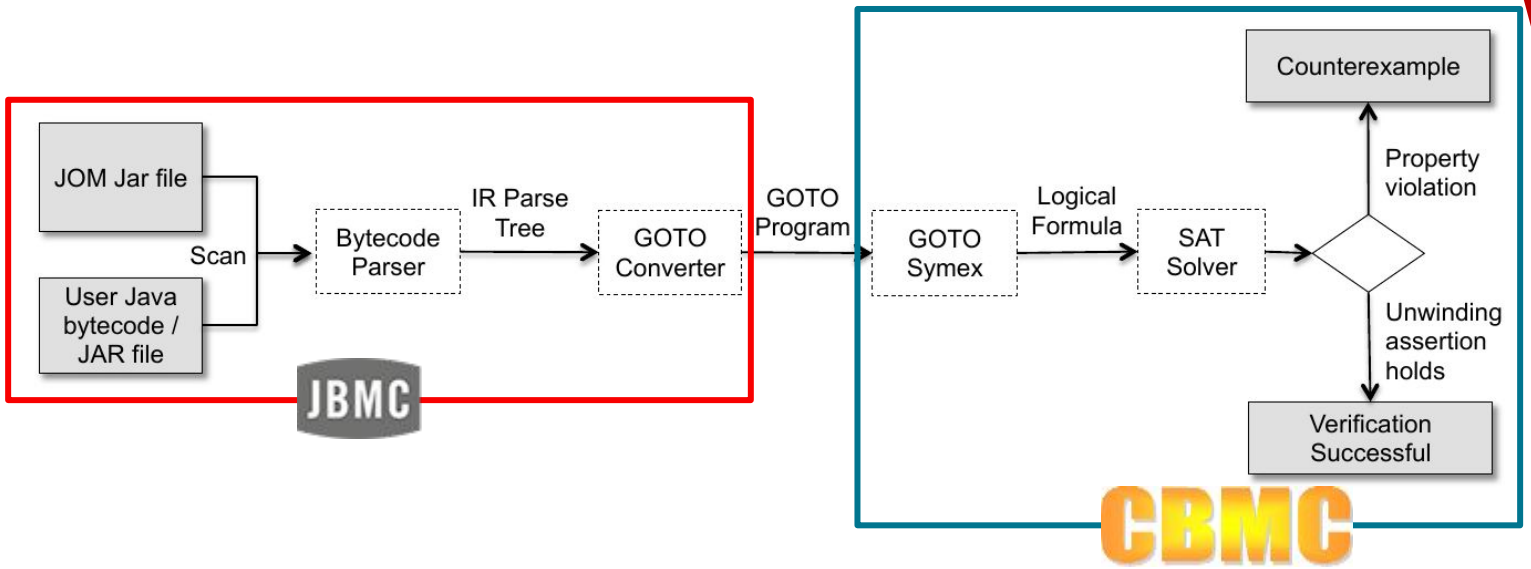
- Object oriented software!
  - Classes
  - Inheritance
  - Polymorphism
  - Exceptions



2.

Architecture

Internal Architecture



What is JOM? (Java Operational Model)

- Verification-friendly model of the **standard** Java libraries
- Java Standard library rewritten to use **CProver API**, and without optimizations
- Implements **most common classes** from `java.lang` and `java.util`

java_bytecode

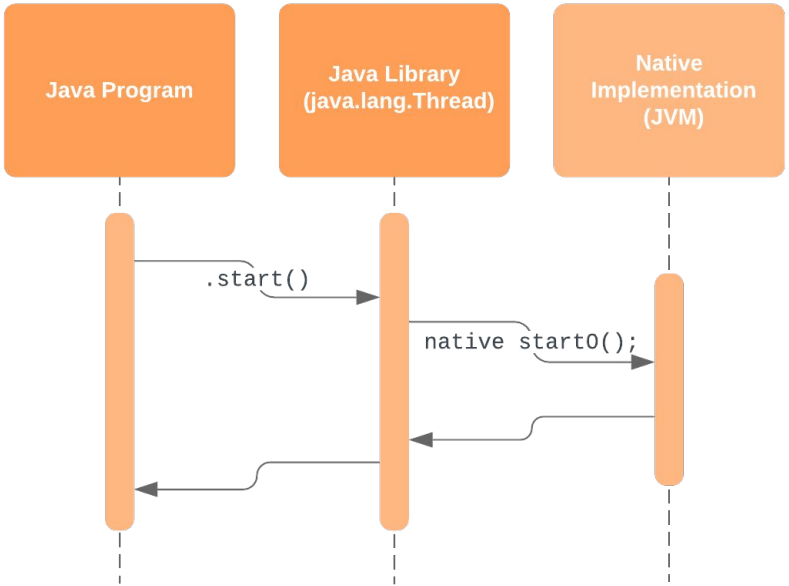
- Translator from Java bytecode to **GOTO code**
- GOTO is a language that can be given to **CProver**
- **Pros**: reuse of the CProver / CBMC stack
- **Cons**: very far away from OO programming; very difficult to translate preserving the semantics; lot of features of Java must be emulated (try/catch, synchronized methods)

How does JBMC use them?

- **No need** to use a specific library, during the compilation the Java library are **dynamically** linked to the program
- During the translation into GOTO code, the module `java_bytecode` **statically links** the method from the JOM
- `CProver.*` methods are **excluded from compilation** and are simply linked to the CProver framework ones (C++ code)

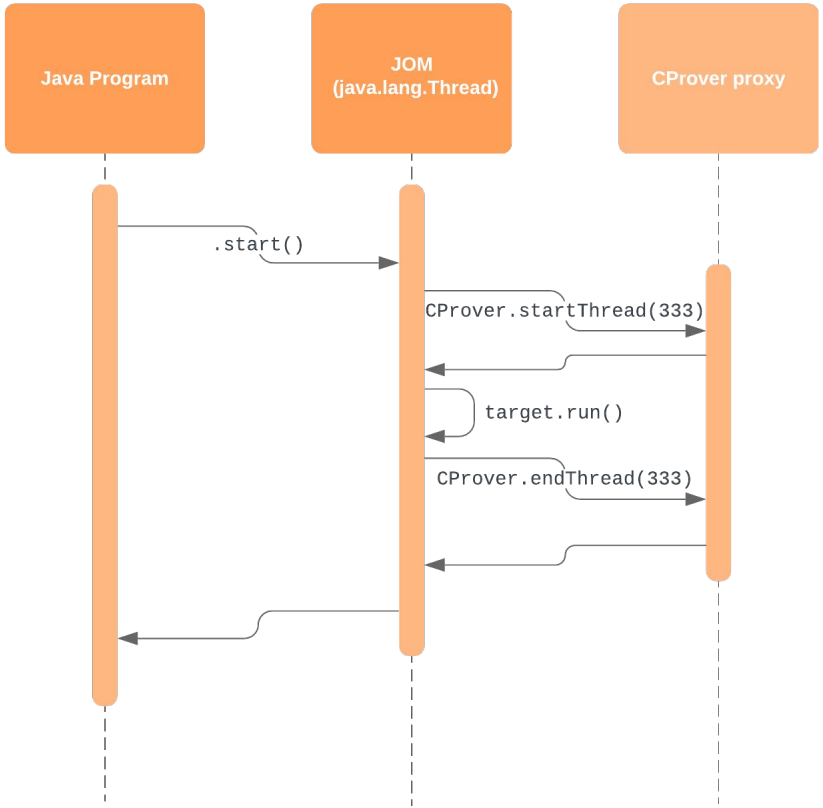
NORMAL EXECUTION

```
new Thread(() -> {....}).start()
```



JBMC COMPILATION

```
new Thread(() -> {....}).start()
```



Removing side effects

```
j = i++;
```

```
j = i;
i = i + 1;
```

Expliciting the control flow

```
while (1) {
    if (x == 200)
        break;
    x = x + 1;
}
y = x;
```

```
while (1) {
    if (x == 200)
        goto while_exit;
    x = x + 1;
}
while_exit: y = x;
```

For to while

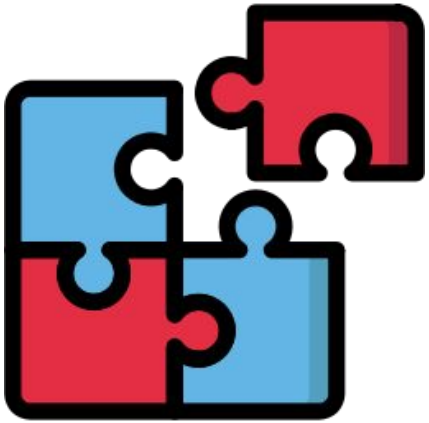
```
for (int i = 0; i < 10;i++)
    j = j * i;
```

```
i = 0;
while (i < 10) {
    j = j * i;
    i++;
}
```

Loop unwinding

```
while (cond) {
    body;
};
```

```
if (cond) {
    // first unwind body;
    if (cond) {
        // second unwind body;
        assert (!cond); // unwinding assertion
    }
};
```



3.

Programming by contract

From JML...

```
1.  /*@
2.  @requires a!=null && a.length>=1;
3.  @ensures a!=null && a.length>=1 ==>
4.  a.length == \old(a).length &&
5.  (\forall int i; 0<=i && i<a.length-1; \exists int
    j; 0<=j && j<a.length;a[j] == \old(a)[i]) &&
6.  (\forall int i; 0<=i && i<a.length-1; @
    a[i]<=a[i+1])
7.  @*/
8.  public void sort(int a[]);
```

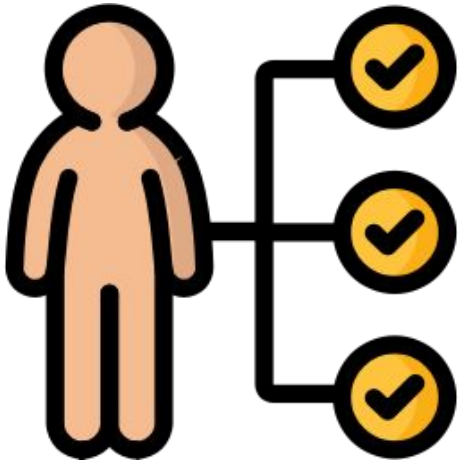
To java verifiable code! (preconditions)

```
public void sort(int a[]){
    assert(a != null);
    assert(a.length >= 1);
    [...]
}
```

To java verifiable code! (postconditions)

```
public void sort(int a[]){
    int[] oldA = Arrays.copyOf(a, a.length);
    [...]
    assert(oldA.length == a.length);
    for(int el: oldA){
        assert(contains(a, el));
    }
    assert(sorted(a));
}
```

VERIFICATION !== TESTING



4.

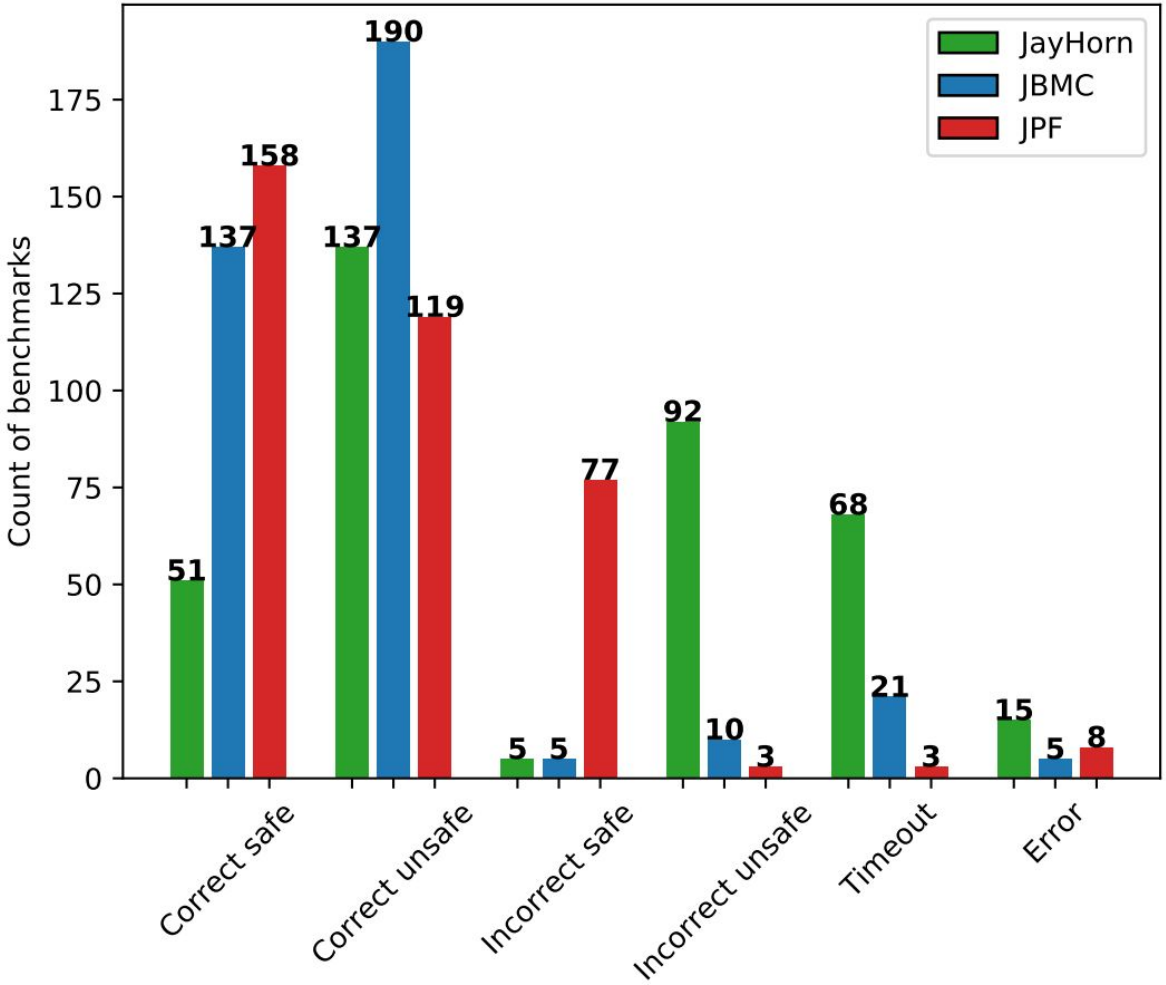
Examples

5.

Pros / Cons

Pros

- Used to verify code, no modelization required
- Usually assertions are not enabled at runtime
  - Assertions check at runtime must be enabled with the option `-ea`
- Checks a lot of common pitfalls by default (array out of bounds etc...)
- Easy to integrate in production workflows (can output `.json` or `.xml` file formats)


JayHorn
JBMC
JPF
Count of benchmarks
0
25
50
75
100
125
150
175
51
137
158
137
190
119
5
5
77
92
10
3
68
21
3
15
5
8
Correct safe
Correct unsafe
Incorrect safe
Incorrect unsafe
Timeout
Error

Cons

- Missing support for most of the java api (eg. Regex)
- Not easy to deduce counterexamples for non trivial programs (but optionally graph is generated)
- Non negligible amount of false positives / negatives for non trivial programs

Thank you for your attention!