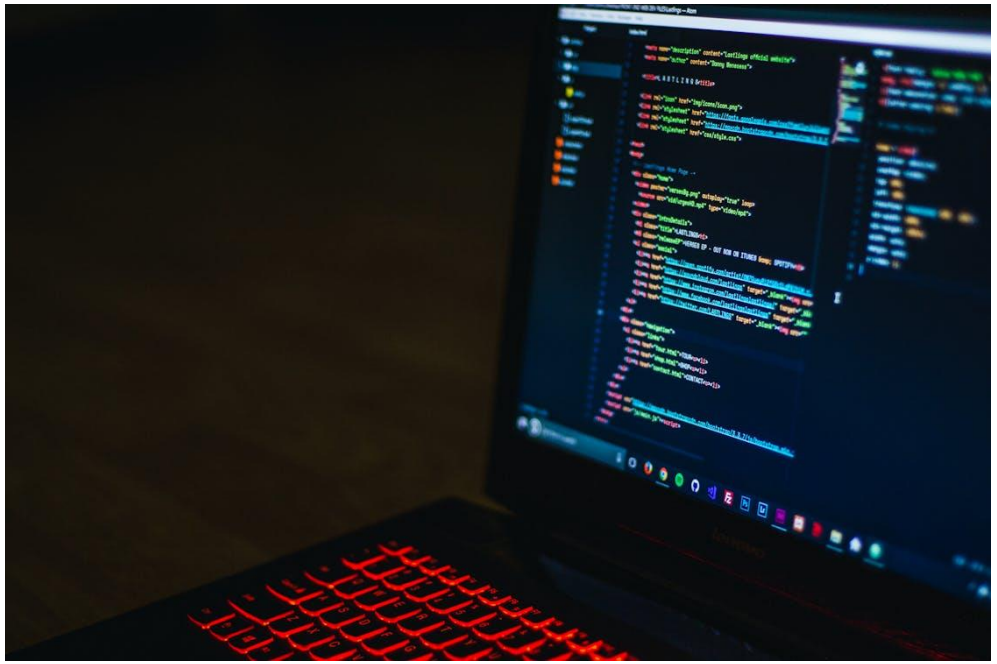
Improving Safety and Security using Static Analysis



Java User Group, 2024

Pascal Kesseli

Background

- Systems Verification Group @ University of Oxford
- Research
  - Static Analysis
  - Program Synthesis
- Open Source Contributions
  - CProver Suite
    - Model Checkers: CBMC / JBMC
  - Mariana Trench

Agenda

- Challenges in Software Testing
- Static Analysis: A Primer
- Exploring different Techniques
  - Linting
  - Model Checking
  - Abstract Interpretation
  - Symbolic Model Checking
- Demo
  - Mariana Trench
  - JBMC

Goals

- Know about different static analysis approaches
  - Strengths
  - Weaknesses
- Know examples of open source tools
  - Security
  - Safety
- Know the trade-offs / cost when using these tools

Challenges in Software Testing

- How do we ensure Software Quality?
  - Code Reviews
  - Unit Tests
  - Fuzzing
- Software Bugs can have massive impact
  - Ariane 5 Rocket Explosion
  - Therac-25 Radiation Overdose
  - Mars Climate Orbiter Loss
  - Knight Capital Group's Trading Glitch
- How can Static Analysis help?
  - Interesting Safety / Security Properties?
  - For these examples?

Static Analysis: A Primer

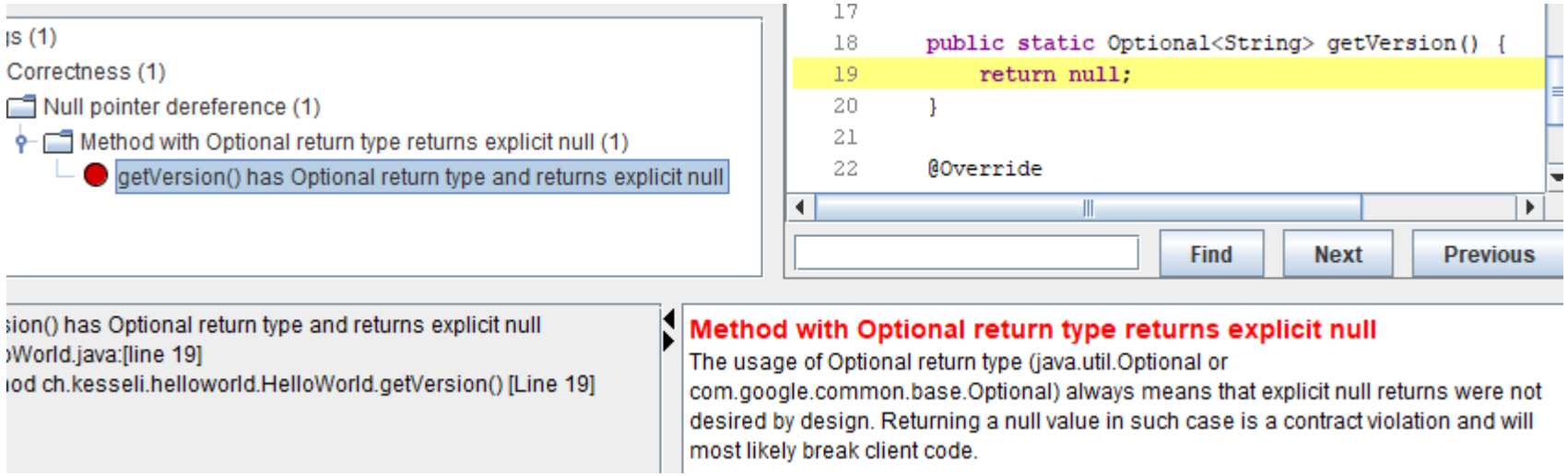
- Analysis of programs without executing them
  - Not limited by runtime constraints
- A Story of Precision
  - Linters
  - Abstract Interpreters
  - Model Checkers
- Terminology
  - Soundness / Completeness
  - Approximation
    - Overapproximate / Underapproximate
  - Syntax / Semantics

Exploring different Techniques

Name	Approach	Properties	Precision	Runtime
Linting	Pattern Matching	• Bad practice • Common errors	Low	Fast
Model Checking	Efficiently explore program states in Custom VM	• General purpose	High	Very Slow
Abstract Interpretation	Explore abstract program state w.r.t. propety	• Specific properties	Medium	Medium
Symbolic Model Checking	Interpret program as formula	• General purpose	High	Slow

Linting

- Example: SpotBugs
  - https://github.com/spotbugs/spotbugs
- Syntactic Analysis
- General purpose
- Pattern Matching over AST
- Data Flow Analysis
- User Annotation Support
  - JSR 305: @Nullable



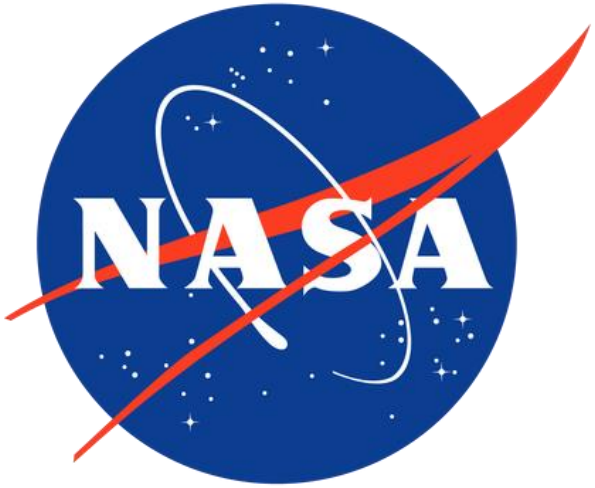
Linting

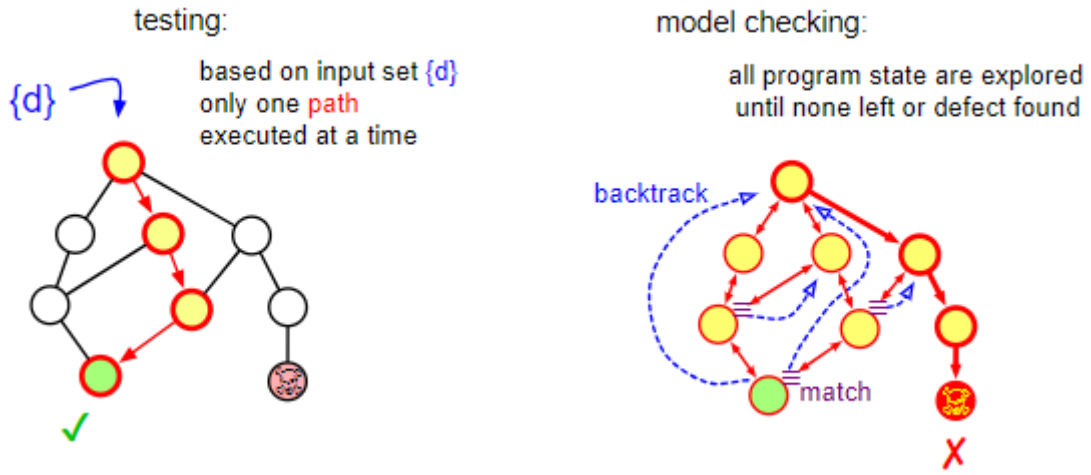
- Fast, but low precision

```
enum Type {
    A,
    B
}

Optional<String> getVersion(Type type) {
    switch (type) {
        case A:
            return Optional.of(value:"1.0.0");
        case B:
            return Optional.of(value:"2.0.0");
    }
    return null;
}
```

Model Checking

- Example: NASA Java PathFinder (JPF)
  - https://github.com/javapathfinder/jpf-core
- Semantic Analysis
  - Common Bugs
  - Custom Assertions / Properties
- Virtual Machine
  - Control e.g. thread scheduling
  - Comparison to Fuzzing
- Exhaust entire state space of the program
  - State space explosion
  - Heuristics
  - Pruning
  - Abstraction
- High precision, high runtime





Model Checking

- State Exploration vs. fuzzing
  - Skip States that we have explored before
- Example:
  - Entry point foo
  - 2^{32} possible input values for i
  - Assuming i is unused after invoking bar
  - For every $i < 0$, we end up in the same state
  - $2^{31} - 1$ possible values for $result$

```
void foo(int i) {
    int result = bar(i);
    // ...
}

int bar(int i) {
    if (i < 0) {
        return 0;
    }
    return i;
}
```

Model Checking

- Abstraction
  - Reason over less precise version of state space
  - Abstract Domains
- Example:
  - Interval Domain
  - Model each number as an interval
  - Apply operations to interval
- Reason over multiple concrete states at once
- Reduces precision

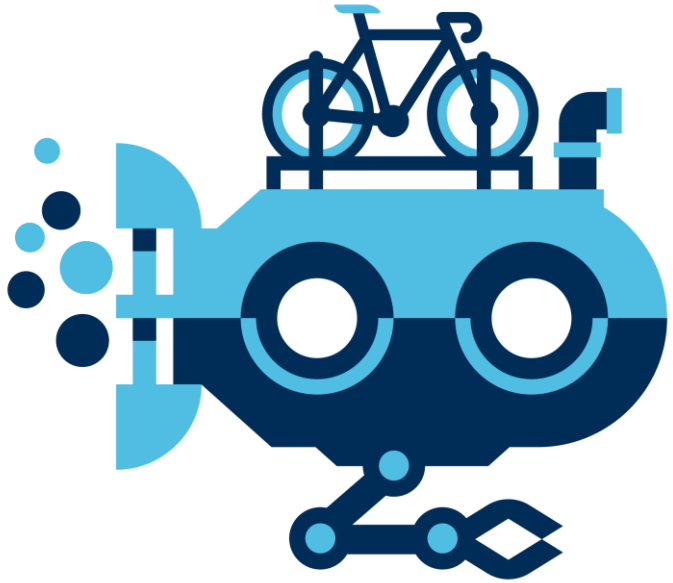
```
boolean foo(int i) {
    // [-2^31 ; 2^31 - 1]
    if (i < 10 || i > 1000)
        return false;

    // [10 ; 1000]
    i *= 2; // [20; 2000]
    if (i == 1001) {
        // [1001 ; 1001]
        return true;
    } else {
        // [20 ; 2000]
        assert i != 1001;
        return i % 2 == 0;
    }
}
```

Model Checking

- Main Takeaways
  - State Space Explosion Problem
- Optimisations
- Abstractions and trade-offs
  - False Positives (Safety)
  - False Negatives (Security)

Abstract Interpretation

- Example: Mariana Trench by Meta
  - https://mariana-tren.ch/
- Feature-rich Abstract Interpreter
- Focussed on Security Properties
  - Flow analysis
  - E.g. Injection attacks
  - Source, Sanitizer, Sink
- Taint Tree Domain
  - Objects (and their fields) have taint «labels»
- Aimed at Security Engineers



Abstract Interpretation

- Injection attacks
- Tainted source (e.g. user-controlled string)
- Reaches vulnerable sink (e.g. SQL database API)

```
int myGet(String userName) throws SQLException {
    String query = "select age from users where name = " + userName;
    Statement stmt = conn.createStatement();
    return stmt.executeQuery(query).getInt(columnIndex:0);
}
```

Abstract Interpretation

- Composition

```
// source: userName
int myGet(String userName) throws SQLException {
    String query = makeQuery(userName);
    return mySqlHelper(query).getInt(columnIndex:0);
}

// sink: query
ResultSet mySqlHelper(String query) throws SQLException {
    // ...
    Statement stmt = conn.createStatement();
    return stmt.executeQuery(query);
}

// propagation: userName -> return
String makeQuery(String userName) {
    return "select age from users where name = " + userName;
}
```

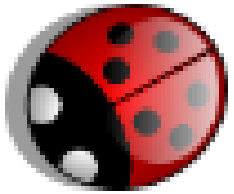
Abstract Interpretation

- Loops / recursion
  - Widening / narrowing: Fixed Point
- Explores CFG
- False Positives
  - Disregards guards
  - Does not model actual values
- False Negatives
  - Maximum depth
  - Polymorphism limit, ...

```
int myOtherGet(String userName) throws SQLException {
    if (!userName.isEmpty())
        userName = sanitise(userName);

    // ...

    if (userName.length() > 0) {
        String query = "select age from users where name = " + userName;
        Statement stmt = conn.createStatement();
        return stmt.executeQuery(query).getInt(columnIndex:0);
    }

    return -1;
}
```

Symbolic Model Checking

- Example: JBMC
  - https://www.cprover.org/jbmc/
- Custom Assertions
- Common Errors (e.g. uncaught exceptions)
- Mathematically reason over programs
- Usually: Map program to a formula
  - Usually SAT or SMT
- Program has a bug iff formula is satisfiable



Symbolic Model Checking

- How are programs mapped to SAT formulas?
- Bounded
  - Loop unwinding
- Single Static Assignment
- Bit Blasting

```
byte foo = 7;
byte bar = 10;
bar = foo;
```

$$\neg foo_0^7 \wedge \neg foo_0^6 \wedge \neg foo_0^5 \wedge \neg foo_0^4 \wedge \neg foo_0^3 \wedge foo_0^2 \wedge foo_0^1 \wedge foo_0^0 \wedge$$
$$\neg bar_0^7 \wedge \neg bar_0^6 \wedge \neg bar_0^5 \wedge \neg bar_0^4 \wedge bar_0^3 \wedge \neg bar_0^2 \wedge bar_0^1 \wedge \neg bar_0^0 \wedge$$
$$(bar_1^7 \wedge foo_0^7) \vee (\neg bar_1^7 \wedge \neg foo_0^7) \ldots$$

Symbolic Model Checking

- SAT Solvers Review
- Solve 3-SAT problem
- NP-complete
- Worst Case: Runtime exponential in size of the formula
  - I.e. The size of the program (including unwound loops)
  - Example: Cryptographic problems

Summary

- All is well then, right?
  - Security Engineer? Use Mariana Trench.
  - Safety Tester? Use JPF or JBMC.
- Open Source Tools
  - Maintenance of specifications, sources, sinks
  - Fixing performance issues
  - Development on the OSS tools used
  - Missing features
  - Missing language support
- Open Source collaboration

References

- Daniel Kroening, AWS
  - Fellow at Magdalen College, University of Oxford
  - Original author of CBMC/JBMC
  - https://www.kroening.com/
- Peter O'Hearn, Lacework Ltd.
  - Professor of Computer Science at UCL
  - http://www0.cs.ucl.ac.uk/staff/p.ohearn/
- Cristina David
  - Senior Lecturer at University of Bristol
  - https://www.bristol.ac.uk/people/person/Cristina-David-d78c4612-1820-443c-b2cb-9db853867d90/
- Lucas Cordeiro
  - Professor of Computer Science at University of Manchester
  - https://research.manchester.ac.uk/en/persons/lucas.cordeiro