Nullness_Lite

An unsound option of the Nullness Checker with fewer false positives

Team Members:
Anny Kong(yk57), Spencer Huang(yh73), Alice Zhao(zhaox29), Cassie Chen(chenm32)

Motivation

- NullPointerException: the most frequent bug in Java Program [1]
- Nullness Bug Detector: a huge convenience for developers

Related Works - NullAway

- Fast: the build time overhead of running NullAway < 10% [2]
- Easy to Use: plugin of ErrorProne which is widely used in industry
- Limitations: does not check code using generics and null assertions [3 & 4]

Related Works - IntelliJ and Eclipse

- Statically checks errors when users type

- “Infer Nullity”: IntelliJ automatically introduces @Nullable and @NotNull to project

Related Works - FindBugs

- Uses heuristic pattern-matching to analyze

- Powerful: directly analyzes bytecode (does not need source code)

Related Works - The Nullness Checker

- A sound, pluggable type checker which aims to find all nullness bugs (type-based dataflow analysis) [5]
- Ground truth of our evaluation due to its soundness
- Conservative: more false positives found [6]

Our Approach - The Nullness_Lite option

- Base: the Nullness Checker

- An unsound option with

    1. Fewer annotations
    2. Fewer false positives

- Provide an upgrade path for users to graduate to the full Nullness Checker

Implementation

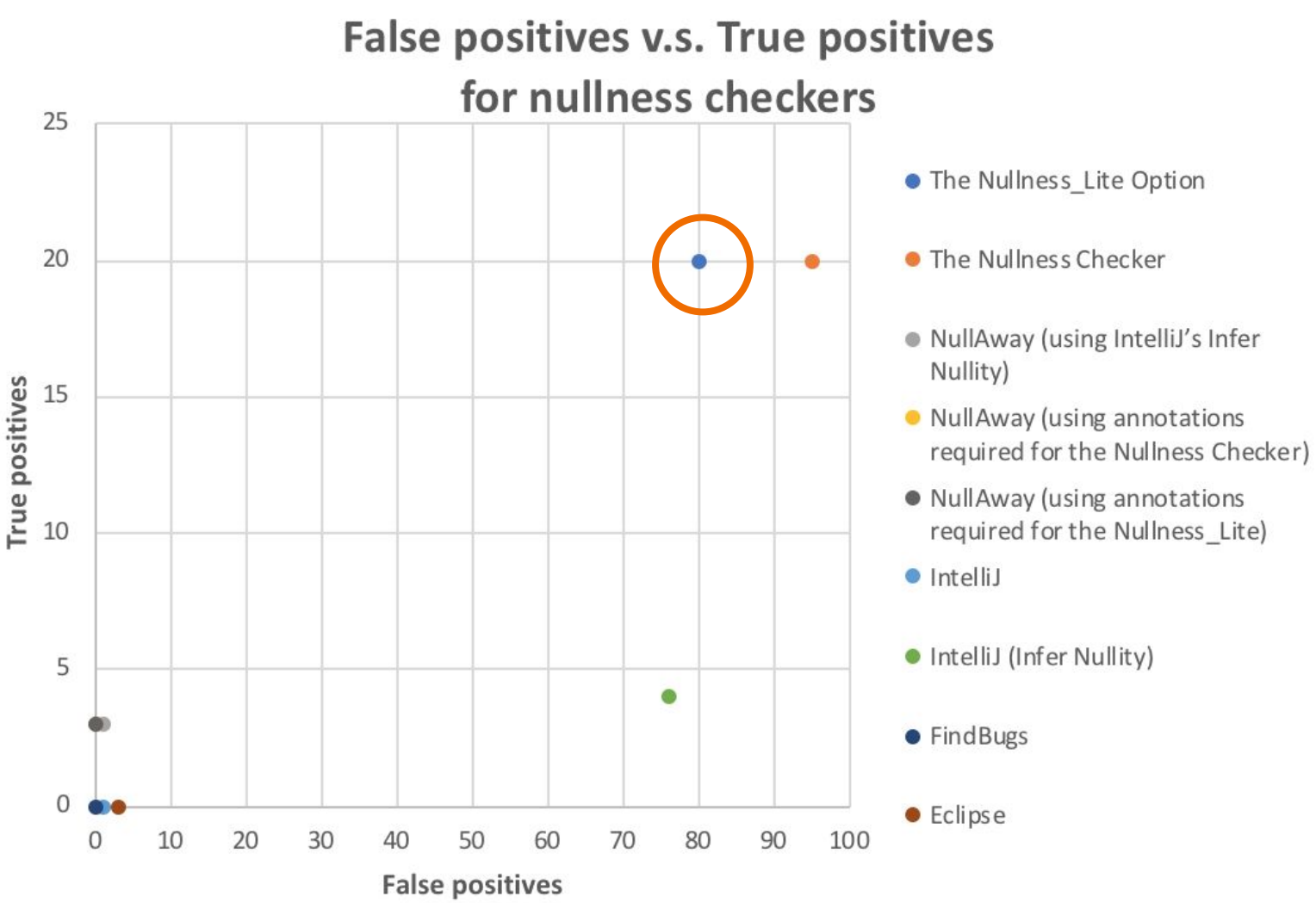
By disabling suggested* features in the Nullness Checker, the Nullness_Lite option assumes

1. All variables are initialized**
2. Map.get(key) returns @NonNull**
3. No aliasing and all methods are @SideEffectFree**
4. BoxedClass.valueOf() are @Pure**

* ideas listed in the Nullness bug detector section of GSoC ideas 2018 [7]
** Refer to our report for details

Command_Line Interface

```
javac -processor nullness -ANullnessLite <MyFile.java>
```


False positives v.s. True positives for nullness checkers
True positives
False positives
0
5
10
15
20
25
0
10
20
30
40
50
60
70
80
90
100
The Nullness_Lite Option
The Nullness Checker
NullAway (using IntelliJ's Infer Nullity)
NullAway (using annotations required for the Nullness Checker)
NullAway (using annotations required for the Nullness_Lite)
IntelliJ
IntelliJ (Infer Nullity)
FindBugs
Eclipse

Evaluation Result

The project we evaluate on: JUnit4

Checkers	True Positives Detected	True Positives Not Detected	False Positives	Annotations Added
Nullness_Lite	20	0	80	318
Nullness Checker	20	0	95	319
NullAway (Infer Nullity)	3	17	1	1285
NullAway (Nullness Checker's annotations)	3	17	0	319
NullAway (NullnessLite's annotations)	3	17	0	318
FindBugs	0	20	0	0
IntelliJ1	0	20	1	0
IntelliJ2 (Infer Nullity)	4	16	76	16*
Eclipse	0	20	3	0

* Since we are evaluating junit4 based on the annotations added by IntelliJ's Infer Nullity, 16 is the number of annotations we changed rather than added. The number of annotations added by Infer Nullity is 1116.

Evaluation Methodology for JUnit4

1. For the Nullness_Lite option and the Nullness Checker, we **add annotations manually to reduce false positives.**
2. An error is an true positive if
    a. the JUnit4 API provides users with functions to raise NPEs
    b. the internal implementation raises NPEs
3. An error is an false positive otherwise

Analysis Examples in JUnit4 - True Positive

```
public static Description createTestDescription(
        @Nullable Class<?> clazz, @Nullable String name) {
    return new Description(clazz, formatDisplayName(name, clazz.getName()));
}
```

The example above is cited from
`src/main/java/org/junit/runner/Description.java`

*Refer to our report and repository for details and more true positive examples.

Analysis Examples in JUnit4 - False Positive

```java
private static final Class<?> MANAGEMENT_FACTORY_CLASS;

static {
  Class<?> managementFactoryClass = null;
  try {
    managementFactoryClass = Classes.getClass("java.lang.management.ManagementFactory");
  } catch (ClassNotFoundException e) {
    // do nothing, managementFactoryClass will be none on failure
  }
  MANAGEMENT_FACTORY_CLASS = managementFactoryClass;
}
```

The example above is cited from
`src/main/java/org/junit/internal/management/ManagementFactory.java`

*Refer to our report and repository for details and more false positive examples.

Lessons learned & Future Work

1. Analyze other features in the Nullness Checker to see whether it is possible to further reduce false positives
2. We can extend our evaluation on these checkers by analysing on more larger, well maintained projects.

Thank you !

Works Cited:

1. Maciej Cielecki, Je ̨drzej Fulara, Krzysztof Jakubczyk, and Jancewicz. Propagation of JML non-null annotations in java programs. In Principles and practice of programming in Java, pages 135–140, 2006.
2. Uber. “Uber/NullAway.” GitHub, github.com/uber/NullAway.
3. Sridharan, Manu. “Support Models of Generic Types · Issue #54.” GitHub, Uber/NullAway, 6 Nov. 2017, github.com/uber/NullAway/issues/54.
4. kevinzetterstrom. “Support for null assertions · Issue #122 · Uber/NullAway.” GitHub, github.com/uber/NullAway/issues/122.
5. Dietl, Werner, et al. “Building and Using Pluggable Type-Checkers.” Proceeding of the 33rd International Conference on Software Engineering - ICSE '11, 2011, doi:10.1145/1985793.1985889
6. Dietl, Werner, and Michael Ernst. “ Preventing Errors Before They Happen The Checker Framework.” Preventing Errors Before They Happen The Checker Framework, www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=7&cad=rja&uact=8&ved=0ahUKEwiup6_V5bPaAhXKrFQKHWgCxYQFghaMAY&url=https%3A%2F%2Fstatic.rainfocus.com%2Foracle%2Foow17%2Fsess%2F1492901668615001brln%2FPF%2F2017-10-02%2520CF%2520%40%2520JavaOne_1507012791774001WJ2t.pdf&usg=AOvVaw3mAtzExTzYm6gr3sCn0cXb.
7. “GSoC Ideas 2018.” Checker Framework Organization: GSoC Ideas 2018, rawgit.com/typetools/checker-framework/master/docs/developer/gsoc-ideas.html.
8. https://github.com/weifanjiang/Nullness_Lite/blob/master/reports/week10/Report-10.pdf