Nullness_Lite

Members: Anny Kong(yk57),
Mengxing Chen(chenm32),
Yuqi Huang(yh73),
Xinrong Zhao(zhaox29)

Motivation

$312 billion per year global cost of software bugs (2013)
$300 billion dealing with the Y2K problem

$440 million loss by Knight Ca
minutes in August 2012
$650 million loss by NASA Mar
conversion bug
$500 million Ariane 5 maiden
16-bit conversion bug

1997: **225 deaths**: jet crash caused by radar software
1991: **28 deaths**: Patriot missile guidance system
2003: **11 deaths**: blackout
1985-2000: **>8 deaths**: Radiation therapy

2011: Software caused 25% of all medical device recalls

- Null -> NPEs
  - Significant
  - Costly - Tony Hoare, “the billion-dollar mistake”
- Current solutions
  - JAVA’s type system
    - too weak
  - FindBugs
  - NullAway
    - no support for assert statement and generics
  - Nullness Checker of the Checker Framework

	Null pointer errors		False warnings	Annotations written
	Found	Missed		
Checker Framework	9	0	4	35
FindBugs	0	9	1	0

support models of generic types #54
Open msridhar opened this issue on Nov 6, 2017 · 1 comment

Support for null assertions #122
Open kevinzetterstrom opened this issue on Feb 13 · 0 comments

Our Approach-->Nullness_Lite

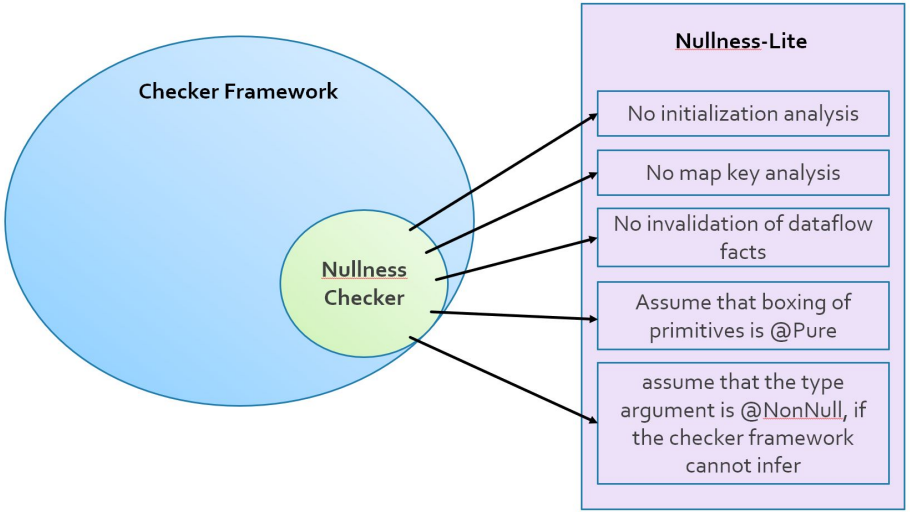
- We decide to build a lite version of Nullness Checker of Checker Framework.
  - Why based on Nullness Checker?
    - It is a stronger analysis tool than other checkers.
      - Guarantees finding all nullness errors
    - Support Generics types and null assertions
  - What is a lite version?
    - less annotations -> easy to use
    - less false warnings -> fast + easy to use
  - Tradeoff?
    - Unsound - does not prevent all possible NPEs

Overall, It catches most of the NPEs with less false warnings while imposing a reasonable annotation burden.

To what extent can Nullness_Lite be implemented to be fast and easy-to-use ?

- **Fast: i.e. less learning, setup and analysis time**
- **Easy to use: i.e. fewer annotations**

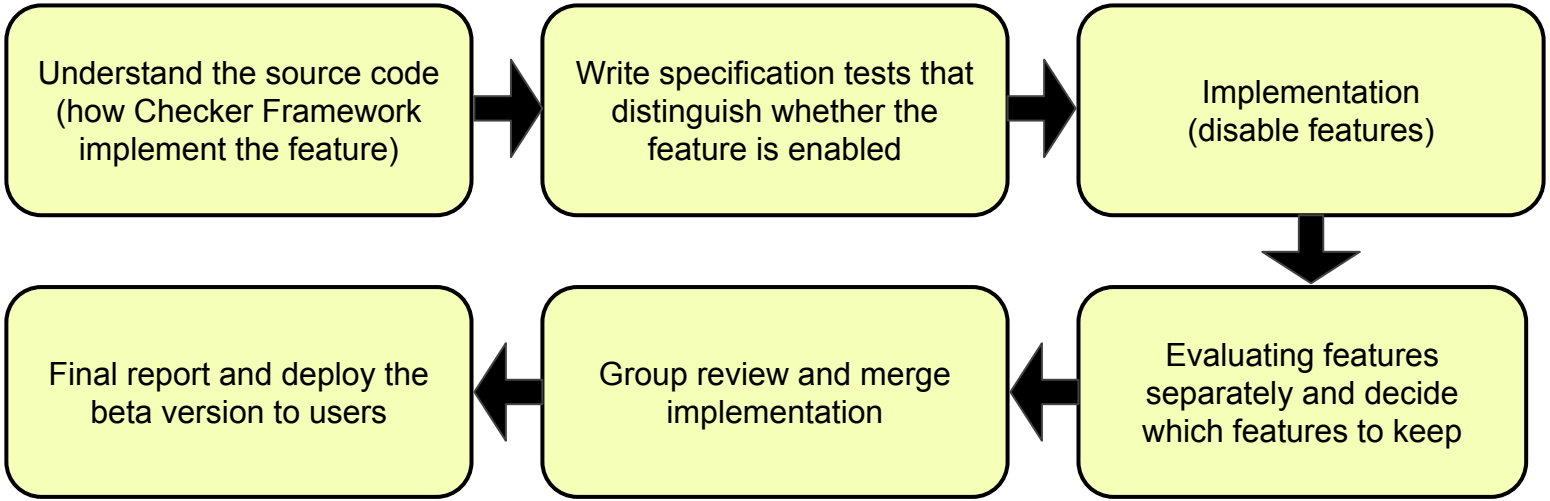
Methodology



- Modifying five features
  - Each assigned to a team member
  - Analyze each feature independently
  - Evaluate the tradeoffs
- Further --Consider more features if time allows
  - Analyze and evaluate each feature of the current Nullness Checker
- Command line interface -- for example

```
zhaox29@ubuntu:~/jsr308/checker-framework$ javac -processor org.checkerframework
.checker.nullness.NullnessChecker -ANullenssLite <your test file>
```

Implementation Plan



Evaluation Plan

- Experimental Subjects:
  - Nullness_Lite, Nullness Checker, and checkers that can detect nullness bugs
- Evaluation for Soundness:
  - Build short tests focus on different features (Nullness Checker has some tests).
  - Manually analyze the tests to get the expected reports.
  - Run tests on each of the checkers, recording the actual true positives and false positives they produced.
- Evaluation for Time Consumed:
  - Find 4 programs with 10k+ lines of source code (bonded with null & using annotations)
  - Run programs on each of the checkers, recording the average time of checking per program.
- Evaluation for # of Annotations Used:
  - Primary: Developers marked down the unused annotations related to the feature they implemented.
  - Subtract from the original amount of annotations used in Nullness Checker.

Checkers Name	Feature to Be Tested	Types of Bugs Revealed (true positives)	Types of Bugs Missed	False Positive (Warnings)	Number of Annotation Used	Average Time to Check X Programs (X = 4)
Nullness_Lite	Features we decide to have					
	No Initialization Checker					
	No Map Key Checker					
	No Invalidation of Dataflow					
	Assume boxing of primitives is @Pure					
	Default inference is @NonNull					
Nullness Checker						
NullAway						
IntelliJ						
FindBugs						
Other checkers						

Preliminary Results

- Our current status?
  - Begin implementation by disabling a sample feature together
- Challenges & Solution
  - Fully understanding Checker Framework
    - In the first feature we disable, we go through the source code of Checker Framework together
  - Evaluate and merge each person’s work
    - Assign group review
    - Analyze the correlated part of each other’s work
      - Shared annotations analysis