NULLNESS ANNOTATIONS AND REFRACTING WITH JAVA

Jiwoong Youn, Devesh Jhunjhunwala, Students; Mohsen Vakilian, Mentor
University of Illinois, Urbana – Champaign

ABSTRACT

Refactoring is a very useful practice which every programmer should integrate within their code development cycle. Refactoring refers to systematically reorganising code to improve readability and structure of the code. This makes things like finding bugs and adding new functionality much easier.

Annotations give us a way to analyze code based on object properties like Nullness and enable us to identify and flag runtime errors during compilation of the code. Specific compiler tools like the Checker Framework, which is programmed to automatically analyze annotated code, generate compile time warnings based on these annotations.

The goal of our research is to understand these annotations and compare the inference tools by applying them to benchmark programs and analysing their results.

CONTACT

Devesh Jhunjhunwala
Jiwoong Youn
University of Illinois, Urbana - Champaign
Email: jolden2@illinois.edu
youn8@illinois.edu

REFERENCES


ABSTRACT

- The annotations we explored were Nullness and Reference Immutability.
- Nullness annotations were inserted into the jolten source code with the help of the inference tool Julia and Nit. The three most important kinds of nullness annotations for an object are Nullable, NonNull and PolyNull.
- Reference Immutability annotations were inserted into the jolten source code with the help of the inference tool Relim. The three kinds of reference immutability annotations for an object are Mutable, ReadOnly, and PolyRead.

CODE SAMPLE 1

Forward is declared as a Nullable array of Nullable Villages.

- Field 'back' must be Nullable since it is being assigned a Nullable value 'y'.

ANALYSING ANNOTATIONS

- We used the benchmark suite of programs, Jolden, to insert annotations and analyse how they affected the code.
- Once the annotations were inserted, we ran the checker framework on each program. By analysing the warnings generated, we gained an understanding of how the checker framework used these annotations to verify object properties.

RESULTS

- The Annotations generated warnings during compilation.
- These warnings were very helpful in flagging possible cases of Null Pointer Exceptions in the code.
- Some warnings were also generated incorrectly.

CONCLUSIONS

- We have studied two different types of annotations which verify the nullness and reference immutability of objects in programs.
- The existing inference tools and the checker framework are not very accurate and still have a few bugs which need to be fixed.

FUTURE WORK

- Our next step is to develop refactoring tools for eclipse which will infer annotations interactively to include the programmer’s insight. This allows him to add/remove annotations in a more controlled environment.

TYPES OF ANNOTATIONS

- The annotations we explored were Nullness and Reference Immutability.
- Nullness annotations were inserted into the jolten source code with the help of the inference tool Julia and Nit. The three most important kinds of nullness annotations for an object are Nullable, NonNull and PolyNull.
- Reference Immutability annotations were inserted into the jolten source code with the help of the inference tool Relim. The three kinds of reference immutability annotations for an object are Mutable, ReadOnly, and PolyRead.

CODE SAMPLE 3

- List is annotated as Nullable.
- Object o is not modified by this method, it is annotated as ReadOnly.

CODE SAMPLE 2

- List is annotated as Nullable.
- Object o is not modified by this method, it is annotated as ReadOnly.

ANALYSING ANNOTATIONS

- We used the benchmark suite of programs, Jolden, to insert annotations and analyse how they affected the code.
- Once the annotations were inserted, we ran the checker framework on each program. By analysing the warnings generated, we gained an understanding of how the checker framework used these annotations to verify object properties.

REFERENCES
