Wesley Cox - coxw2
Grant Hughes - gyhughes
Joe Santino - jsantino
Team Website: http://students.washington.edu/gyhughes/cse403/TeamWebsite/

Index Checker

Product Description

Product Overview

The Index Checker will allow developers to annotate their programs with new types to avoid Index Out OF Bounds exceptions at run time. Instead when they run the checker over their program it will issue warning about places that are unsafe. Thus enabling developers to address these issues by either fixing the unsafe access, or if they believe it is safe, suppressing the warning. When used correctly and not suppressing any warnings that are actually unsafe, The Index Checker can guarantee that none of the array accesses (and later list accesses) will cause an Index out of bounds exception at runtime.

The index checker is a new checker built within the Checker Framework[1]. The checker framework currently supports adding type annotations to certain variables to issue warnings for code that could throw a runtime exception.

Our plan is to make the type checker as simple and as similar to possible to the default type checkers provided in the Checker Framework. Our current plan is to require developers to do the following to use our type checker:

1. Obtain the Checker Framework for the developer's environment.
2. Obtain the Index-out-of-bounds typechecker
3. Annotate desired indices with @IndexFor

Alternatives

FindBugs is tool for java development that looks for bugs by doing static analysis on compiled Java byteCode. It does by default contain checks for "Array index is out of bounds" bugs. This program works by looking for bug patterns, or code idioms that are often errors [2]. Our project is different than FindBugs because instead of looking for patterns, it will use static analysis to generate warning. From the user's point of view, warnings that are issued will be more consistent, both in correct and incorrect warnings.

Currently, It is very difficult to find a programming language that contains a typechecker that catches index-out-of-bounds errors at compile time. For pluggable frameworks such as Checker Framework or JavaCOP, It appears that no pluggable type system has been implemented to check for index-out-of-bounds errors. We do not have any evidence to prove or refute this, as the topic of checking for index-out-of-bounds errors is currently an unpopular topic. The general consensus of the community from many non-scientific programming websites (such as Stack Overflow) is that although trivial index-out-of-bounds errors are easy to catch, dynamic array size allocation and similar popular practices are too difficult to catch perfectly. To cope with this, we simply do not promise that false positives and true negatives will not exist. This is why we will only issue suppressible warnings unlike how Java's type system denies compiling until the errors are fixed.

Major Features

- Provide an @IndexFor(array[]) annotation that when properly used can guarantee there will be no index out of bounds exceptions at runtime.
- Issue Warnings for potentially unsafe array accesses.
- Issue Warnings for potentially unsafe List accesses.
- Provide functionally to automatically refine types when developes check bounds of arrays/lists. Leading to less annotating with the same guarantees.

Stretch Features

- Provide annotated versions of some java libraries to create stronger guarantees for developers.
- Compatibility with multi dimensional arrays

Non-functional Requirements

- The Index Checker will be built as an extension to the Checker Framework
- The Index Checker will not cause code that previously compiled to no longer do so once annotated.
- The Index Checker will be compatible with all other standard checkers within the Checker Framework

External Documentation

We are going to provide a user manual. This will take the form of a short 1-2 page document outlining only what a user needs to know in order to use the system.

We will also provide a document showing how the types work when checked including all the backend annotations. Understanding this is not necessary to use the checker but some people may be able to use them in creative ways we haven't anticipated.

Process Description

Toolset

We will be writing our project in Java, this is because java is a widely used language, that all members of the team have a lot of experience in. Also java is the language that the Checker Framework is written in which is the tool we are extending.

Additionally we will be using The Checker Framework, which is an open source tool for creating the type of annotation we are making. We are using this tool because it has all of the functionality we wish to extend and the pluggability we are trying to achieve.

Group Dynamics

Since we have a small group everyone will contribute to each part of the project. However each member of the group will have a specific focus that they will be responsible for taking charge of.

Wesley: Project manager(in charge of organization and coordination)
Grant: Director of Web Management
Joe: Lead Designer

All of these things are ongoing and need to be taken care of for the entire project development. As Such these roles will remain in place for the duration of the development.

These roles were decided as such:
Wesley is our project manager because he has been doing a lot of the coordination so far, looking forward to the end goals of the project the most, currently in charge of finding use cases and case studies so far.

Grant is our Web Manager because he has the most experience with web design. He has already created the website for our team and is currently in charge of maintaining it.

Joe is our Lead Designer because he has contributed the most to our projects design. Having been in charge of UI diagramming and User Manual, he has the most insight about the design of the system and is most capable of changing it further down development.

Basically the roles we chosen based on what the team members focused on most over the first couple weeks. This showed where their interest was and therefore we believe that these roles will keep the team excited for the project and working on what they enjoy most.

Timeline

Aside from this schedule there is maintenance of the web page, redesigns of the system, and coordination towards goal. The web page maintenance will take place as new documents are produced and need to be uploaded. This is Grant's main job aside from development. Additionally as we work we expect there to be times at which some functionality needs to be better supported, or something made less confusing. When these things require redesigning parts of the system this will be taken care of by Joe. lastly there will be points where we come across certain features or functions that we currently don't support or have planned, deciding how we address these and if we want to include them will be up to Wesley.

Week	Plan
April 11 - April 17	All:Requirements specification, trivial checker work Joe: user side class hierarchy, user manual Wesley: find case studies, use cases Grant: create website
April 18 - April 24	All: Finish Trivial type checker (@Zero) (if not complete) Joe: Software Design Specification Wesley/Joe: implementation plan(classes, interfaces) Grant: File and Repo organization
April 25 - May 01	All: Skeleton implementation (Zero -feature) // Ability to add annotations and compile code with the annotation, although they don't do much yet
May 02 - May 08	All: Beta implementation // annotations work to point out possible out of bounds for arrays at compile time (still buggy) // work on list functionality
May 09 - May 15	All: Make sure beta passes all tests Grant: Test more, fix issues, finalize beta release Joe: List functionality Wesley: start case studies
May 16 - May 22	Joe/Grant: Create complete release // Wesley(all once other part done): case study work

May 23 - May 29	All: finish case studies and evaluate effectiveness // code review
May 30 - June 05	All: Final release // demo

Here, we believe that allocation 2-3 weeks for actually writing the project and 2 weeks for debugging is a good schedule to follow, as the code we are writing should be relatively short for a project this size. This leaves us with 2-3 weeks to do the case studies and evaluations, which should be ample time to do a thorough evaluation

Risks

The biggest risk to completing the project right now is learning to use the typechecker framework quickly. None of us have previous experience with type-checking code, so we need to learn what is and is not feasible to sanity check our design. If this can be done quickly enough, we then can spend adequate time actually building our type checker.

We are trying to mitigate this risk by first developing a really trivial type checker to learn the workings of the type checker framework. The plan is to make a rudimentary @Zero checker, which can later be integrated into the complete project.

The checker also has a risk of being too difficult to be added to developed code. If the annotations are really difficult to understand, or take a lot of time add to code developers may not want to use them. To decrease the work it takes to put in the annotations we plan to work on adding useful introduction and transfer rules that can keep a lot of the work behind the scenes. Therefore limiting the manual annotating and making it more likely that developers will use the tool.

Additionally, if the plugin restricts functionality of arrays it could cause problems with usability in some situations. In order to mitigate this risk we will use the case studies to see how well we can annotate without changing functionality. If necessary we could possibly weaken our guarantees (limit to certain situations) in order to warn for some errors while still being useable overall.

Referenced works

[1] http://types.cs.washington.edu/checker-framework/

[2]University of Maryland, ‘FindBugs™ Fact Sheet’. [Online]. Available: http://findbugs.sourceforge.net/factSheet.html. [Accessed:12-Apr-2016].