Nullness Analysis of Java Source Code

Arnout F.M. Engelen

Abstract

Explained in 3 words, this research is about “reducing software crashes”. Besides traditional testing, a more rigorous approach to reducing errors is called `static source code analysis'. Instead of trying to run the program and seeing if a situation can be created where it crashes, static analysis looks at the source code of a program without running it. A static analysis tool knows about certain circumstances that lead to crashes, and verifies that the program will never be able to get in such a state.

One such situation is known as a `null dereference'. Many programming languages support `references' which, when the program is running, can either be `null' or have a value. If such a `reference' is `dereferenced' while it is `null', this can lead to unexpected behaviour and often a crash.

If a static source code checker could be constructed that can find the locations where a null dereference might occur, this class of errors would never again cause crashes in production code.

The good news is such checkers can and have been constructed. Unfortunately, they cannot do it alone: they rely on so-called `annotations' that a programmer can add to his code. Annotations are statements by the programmer about the code, such as “this reference here is never null”. The tool can then check if this is indeed always the case.

This technique works quite well if the programmer supplied these annotations from the start, gradually while writing the code. However, in practice it is often desirable to run the checker over an existing program which does not yet contain any annotations. Manually removing false alarms by adding missing annotations is a tedious and time-consuming task. Because of this it is desirable to have tools that generate some annotations automatically.

This research project investigated the requirements for tools that provide assistance for this task. We have evaluated existing tools, highlighted important design decisions and made recommendations. Finally, we constructed a better Annotation Assistance Algorithm and wrote a proof-of-concept implementation to verify that our approach indeed works in practice.