Funqual - User-Defined, Statically-Checked Call-Tree Constraints in C++

Andrew Nelson
Masters Thesis Draft

February 28, 2018

Abstract

In this paper we create a static analysis tool called funqual. Funqual reads C++17 code "in the wild" and checks that the function call graph rollows a set of rules which can be defined by the user. We demonstrate that this tool, when used with hand-crafted rules, can catch certain types of errors which commonly occur in the wild. We claim that this tool can be used in a production setting to catch certain errors in code before that code is even run.

Contents

1 Introduction **2**

2 Background **2**

3 Design **2**

4 Implementation **2**

5 Methodologies **2**

6 Analysis **2**

7 Future Work **2**

8 Related work **2**

1 Introduction

Writing bug-free software is challenging if not impossible. In the past 30 years, millions of dollars have been invested in tools that help developers write code that is robust, readable, and correct. In general these tools fall into two categories: Dynamic Analysis tools such as gdb, valgrind, and IDA which analyze programs as they are running; and Static Analysis tools such as lint, cppcheck, and GCC. All these tools have different use cases and can be used in conjunction to write code that is error-free.

Languages like C++ and java are well suited for static analysis because type information is readily and epxlicitly available in the source code. In fact, a lot of static analysis happens during the compilation phase. This analysis finds and reports bugs like type mismatches, undefinied references, and conflicting definitions.

The following snippet of code demonstrates this concept:

```
int main(void) {
  int i = 9;
  char *j = "pandas";
  return i + j;
}
```

If C somehow did not perform any sort of static analysis, this would be a runtime error (or worse may fail silently at runtime). GCC, however, checks the types of all operation and operands at build-time in order to build efficient machine code. When compiled with GCC, the code above gives an error about addition between in and char*. While innexperienced programmers may find this knack for finding type-errors to be cumbersome, more experienced programmers often learn to love this static checking that comes for free with the language.

Of course, there are things which are impossible to statically check in C and C++. For years, various projects have tried to build tools which can statically analyze code to check for memory errors, unit errors, and infinite loops. Unfortunately, many of these projects require specific language extensions in order for there to be enough information in the source code for the tools to work. This is unfortunate because it prevents the tools from being used on code "in the wild", or code that has not been written with the tool in mind.

In this paper, we create a tool called funqual which can statically check the call-tree of C++ code "in the wile" for certain patterns and report back to the user. The program uses libclang to parse C++17 code and build a call-graph for the entire program. The program then checks this call-graph against user-defined rules which encode.

2 Background

3 Design

4 Implementation

5 Methodologies

6 Analysis

7 Future Work

8 Related work

References