hbc: A CELEBRATION OF UNDEFINED BEHAVIOR

Andrew Benson*
Department of Computer Science
Stanford University
Stanford, CA 94305
adbenson@stanford.edu

April 1, 2020

ABSTRACT

We present `hbc`, a next-generation compiler with half the performance of `clang` but twice the fun. We prove by way of demonstration that undefined behavior in a spec-abiding C compiler can do more than just segfault.

Keywords Compilers · Undefined Behavior · Bash Misuse · Computer Music · Principles of Imperative Computation

1 Introduction

The C programming language has gone through several iterations (C89, C99, etc), but each edition leaves some behavior under-specified. For example, while the C standards specifies that a plain `char` must be 1 byte in size, it does not specify its sign. Each compiler that implements C may choose whether a plain `char` is signed or unsigned. Because this behavior is not dictated by the standard but implementations must define it in a consistent fashion, this behavior is called **implementation-defined**.

Some other behaviors are even looser. Not only do the C standards not prescribe a particular behavior, but implementers are not required to make the behavior consistent. The most infamous example is probably dereferencing a `NULL` pointer (or any other invalid pointer). The C standards do not say what should happen, and while code produced by most compilers will probably segfault, it does not have to, nor does it have to do the same thing on two separate occasions. It is perfectly valid for it to delete your tax statements, or even to start playing a melody through the speakers! (That's called foreshadowing.) Because no definitions are specified by the C standards or the compilers, this is called **undefined behavior**.

2 Motivation

Computer science students are generally introduced to these concepts in their first systems-level course in C. In the author's experience, students are generally confused by these concepts since mainstream compilers are rather consistent in either hanging or segfaulting when encountering these behaviors. Although the concepts are distinct, students tend to equate "segfaults" with "undefined behavior" and direct the former's rightfully deserved ire onto Poor, Mistreated Undefined Behavior. Not only is there a misunderstanding of what these foundational concepts mean, but students develop a fearful anxiety and distrust of Undefined Behavior. It's unjust, and it's time that someone stood up for Undefined Behavior. The mainstream compilers like `gcc` and `clang` are too self-absorbed in exploiting Undefined Behavior for frivolous goals such as "performance" and "implementation-simplicity", so the only way is to build a new compiler that respects Undefined Behavior for what it is and what its potential can be.

The author also recalls sitting in a 15-122 lecture freshman fall where the professor suggested that a C program could play 'Happy Birthday' during undefined behavior. That may also have had some influence on this project.

*Also a software engineer at Google, which was not involved in this research.

3 Design

And by "new compiler", what the author really means is "wrap an existing compiler (one of those he previously mocked) and call it a new one". This is an application of the Software Engineering Principle "ain't nobody got no time fo dat" [building a compiler from scratch]. The author procrastinated and only had a few hours to build it, okay?

The main goal of this new compiler, `hbc` (the Happy Birthday Compiler), [2] is to compile C programs correctly but for some subset of undefined behavior, play the "Happy Birthday" song. The subset of undefined behavior was chosen to be specifically when a NULL pointer is dereferenced. [3]

It is well-known that the general problem of reasoning about the runtime value of anything in a program reduces to the Halting problem. Thus it's clear that checking for `NULL` pointers cannot be done statically, and null-checks must be inserted into the code. A LLVM compiler pass seems ideal for this, although ironically the pass will clutter the code instead of optimize it. This would be done at the IR level, but we can simply consider LLVM load and store instructions. The injected code can examine the dereferenced value and if NULL, can call our custom code that plays Happy Birthday.

3.1 Getting the Music File

The author doesn't fancy getting sued, so he didn't want to rip a track of Happy Birthday off the Internet. Since this compiler is certainly going mainstream, the costs of a commercial license would also be out of the question. Thus the author painstakingly dragged and dropped sine waves in Audacity and recreated a tasteless rendition of the tune by ear. He hopes that's enough to avoid lawsuits.

3.2 Playing the Music File

It turns out there's no playMyOggVorbisMusicFile() syscall in the Linux kernel. Who knew? The author eventually decided to link against `libcanberra`, a Linux library that is apparently capable of playing audio files.

Using this, we create a C function that takes in a pointer, checks whether it's `NULL`, and if so, plays Happy Birthday. Since all of this has to end up in the compiler's outputted executable, we embed the entire audio file into the C source code as a base64-encoded string. Was this a good idea? The jury's still out.

3.3 The LLVM Compiler Pass

The compiler pass doesn't seem hard - just throw null checks onto every load or store instruction, and pretend that it's not going to destroy performance (it is). We could definitely make this significantly better by doing a dataflow analysis and removing null checks for statically provably non-null pointers (e.g. pointers that have previously been checked), but that would have taken effort.

But it was still difficult for the author because he's bad at C++, especially LLVM's variant.

3.4 Outputting an Executable

Because the author enjoys pain, he decided that the compiler executable would just be a bash script that drives each portion of the compilation process. Yep. At a high level, the script compiles the input C files to LLVM bitcode, links the bitcode into a big bitcode file (along with the bitcode for the function that plays the music file), runs the LLVM pass, assembles the bitcode into assembly, and links it alongside its dependencies.

3.5 Shell Scripting Inception

As briefly mentioned, everything needs to go into a single compiler executable, including the bitcode for the music player function and the shared object implementing the compiler pass. The author brilliantly decided to inline all of that into the shell script, again using base64 encoding. And because he didn't want to re-inline everything very carefully each time any of the inlined dependencies changed, he wrote a bash script that generates the bash script executable that compiles C programs into Happy Birthday-playing executables. If stacking more layers is a legitimate solution in Machine Learning, why not here?

[2] The code for `hbc` can be found at https://www.github.com/anbenson/hbc.

[3] It was chosen because the author ~~is lazy~~ wanted to pick a behavior many programmers have experienced before.

Figure 1: I promise it's a lot more fun when you can hear it playing Happy Birthday.

3.6 Testing

you always pass all the tests you don't write

4 Properties

I claim that `hbc` is a standards-abiding C compiler.

Proof. By reduction.

We assume that `clang` is a standards-abiding C compiler. Let X be an arbitrary C program from the space of valid C programs. Suppose X does not dereference a NULL pointer on any input. Then X's behavior is identical on `hbc` and `clang`. Suppose there exists an input for which X dereferences a NULL pointer. Consider the first instance in time at which X dereferences a NULL pointer. This is undefined behavior, so anything after this point in time should not be considered. But in everything before this instance, X has the same behavior on `hbc` and `clang`. Thus `hbc` has identical behavior to `clang` on non-undefined behavior and thus it is a standards-abiding C compiler.

5 Results

The author tried a test case or two. They seem to work. So it probably works. But you should try it, it's fun to see your program suddenly burst into song when it dereferences a NULL pointer.

The author did not try any benchmarks, but they're probably disappointing.

6 Applications

I dunno. It's fun? It serves as a basis for a SIGBOVIK paper? It's a fun aside in a 15-122 lecture?

7 Future Work

Nope, we're done here.

8 Acknowledgements

The author would like to thank his 15-122 professors, Tom Cortina and Rob Simmons, who were part of the inspiration for this project and helped cultivate an interest in C and compilers.