

Project Proposal: Safe C using Fat Pointers

Miguel Silva
<http://andrew.cmu.edu/~miguel/safec>

March 23, 2009

Project Description

C and other unsafe languages allow programmers to easily access the wrong memory regions and to disregard the type system completely. This enables many security holes and bugs, the most well-known and historic relevant of those being the *array overflow* problem. Nonetheless, C remains one of the most used programming languages, with a large amount of software written in it, and several other languages that depend on its libraries.

In order to solve these issues, we will develop a method that uses *fat pointers* to ensure the code's memory safety. Our work will be based on CCured [3–5] and will apply the techniques used by that compiler to the LLVM IR. Specifically, we will create an analysis based on a trimmed version of the CCured type system and implement the following optimizations:

- **Dynamic Check Placement:** Usually, dynamic checks must be performed whenever a pointer is dereferenced. However, the number of checks can be reduced if, for instance, they are locally redundant or if it is clear that the pointer will be dereferenced at least once (in that case we check when creating the pointer)
- **Forward/Backward Pointers:** Some pointers are only incremented (or decremented) and therefore we only need to keep track of the pointer's upper (or lower) bound.
- **Loop Optimization:** In [6], the author proposes a loop optimization specific to the CCured compiler, that moves some checks to the outside of loops.

Furthermore, we hope to see how the different optimizations already present in LLVM affect our analysis.

Our goal is to create a method that does not require the source code to be rewritten, that works for any language, and that can be used to check if optimizations preserve safety. Additionally, if the code is legal we want it to run

without raising exceptions and without major overhead.

The project's webpage is <http://andrew.cmu.edu/~miguel/safec>

Related work and Literature Search

As previously stated, our approach is similar to the one designed for the CCured [3-5] compiler. The major difference is the fact that CCured requires some code rewriting and is a source-to-source compiler. Furthermore, we will extend the set of optimizations to include the loop optimization presented in [6].

In [7], the author describes a compiler that accepts any unaltered C program and ensures the result is safe. However, this compiler depends heavily on run-time checks and analysis, while our technique will collect the bulk of information statically, and try to reduce the amount of dynamic checks to the minimum, in order to mitigate the overhead.

It is also worth mentioning the existence of safe C dialects, such as Popcorn [2] and Cyclone [1].

Schedule

Getting Started

So far, no actual work has been done, besides reading about CCured and other safe compilers for C. However, we hope to reuse some parts of the code written for the class assignments.

Week 1 and 2	Implementation of an analysis pass based on the CCured type system Implementation of a pass that implements the required dynamic checks
Week 3	Milestone (below)
Week 4 , 5 and 6	Optimization implementation

Milestone

We will use this initial version of our tool to check the ratio of legal programs that raise exceptions and identify common situations that cause problems. We will

then evaluate the compiler in terms of overhead, and make an initial comparison with CCured.

Critical Path

The critical Path in our project will be the implementation of the optimizations. We hope to use the information gathered by evaluating the analysis to improve or design new optimizations.

Software required

In order to evaluate the accuracy and performance of our system, we will need examples of C code that are representative of real programs and common problems. Most of these examples can be found in benchmark suites, such as SPECINT95, Versabench and the Olden Benchmark suite.

References

- [1] Trevor Jim, Greg Morrisett, Dan Grossman, and Michael Hicks. Abstract cyclone: A safe dialect of c.
- [2] Greg Morrisett, Karl Crary, Neal Glew, Dan Grossman, Richard Samuels, Frederick Smith, David Walker, Stephanie Weirich, and Steve Zdancewic. Talx86: A realistic typed assembly language. In *In Second Workshop on Compiler Support for System Software*, pages 25–35, 1999.
- [3] George C. Necula, Jeremy Condit, Matthew Harren, Scott McPeak, and Westley Weimer. Ccured documentation.
- [4] George C. Necula, Jeremy Condit, Matthew Harren, Scott McPeak, and Westley Weimer. Taming c pointers.
- [5] George C. Necula, Jeremy Condit, Matthew Harren, Scott McPeak, and Westley Weimer. Ccured: type-safe retrofitting of legacy software. *ACM Trans. Program. Lang. Syst.*, 27(3):477–526, 2005.
- [6] AJ Shankar. Loop optimization for ccured.
- [7] Oiwa Yutaka. *Implementation of a Fail-Safe ANSI C Compiler*. PhD thesis.