Abstract

The phrase “floating-point number” is a lie. Floating-point arithmetic is fundamentally interval arithmetic wherein the interval lengths are a function of the median point.

We propose a project to develop an automated tool that reasons about the accuracy of floating-point arithmetic. It aims to discover instances where the floating-point arithmetic result is not the most accurate floating-point approximation to the real number implied by the algorithm (if interpreted with real number, instead of floating-point, semantics).

Keywords floating-point arithmetic, bug finding

1. Introduction

The tool will be applied to programs written in the C language that use:

- floating-point numbers,
- elementary floating-point operations, and
- transcendental floating-point operations

There are 8.4 million results for the keyword “float” in the C source code on GitHub. This certainly over counts the number of files using floating-point arithmetic, but we’re confident this is a large class of programs.

We may further divide the class of floating-point-arithmetic-using programs, a non-exhaustive list follows.

- physical and biological simulations
- video games, both for graphics and physics
- signal processing applications, such as image and sound processing
- embedded real-time control systems
- embedded sensor systems

2. The Ideal Product

The ideal product of our research is an LLVM interpreter, modified such that:

- each floating-point number has a real number companion
- floating-point operations (both elementary and transcendental) induce real number operations on the companions
- a warning is issued for each floating-point output that is not the best approximation to its real number companion

Moreover, the interpreter should be sufficiently efficient for use in a major software company’s build cycle. Consider for example Facebook’s standards as described in the INFER paper (Calcagno et al.). Maintaining this performance may require implementing performance heuristics that leverage real number semantics, such as replacing triangle number sums.

Finally, the project will generate a corpus of in-the-wild floating-point arithmetic errors.

3. The Minimal Product

The minimal product of our research is an LLVM interpreter, modified such that:

- every floating-point number has a real number companion
- elementary floating-point operations induce rational number operations on the companions

4. Possibility of Failure

We are most concerned about modifying the LLVM interpreter because we collectively have no experience with LLVM. We are also concerned with:

- finding a sufficient large and interesting corpus of programs
- finding non-trivial, correctness-relevant floating-point arithmetic bugs

5. Technologies

We will not implement a new interpreter nor a new C compiler. We will directly analyze LLVM bytecode, reusing an existing LLVM interpreter. Moreover, we will not implement a new Real number library for C. We will use RealLib, a GNU LGPL licensed library for performing both elementary and transcendental real number operations.

6. Related Work

If a single paper must be chosen, we believe the class as a group of programmers would benefit from reading the first few sections of What Every Computer Scientist Should Know About Floating-Point Arithmetic by David Goldberg (Goldberg 1991).
7. The One-Week Project
Our one week project is the production of a modified LLVM interpreter that uses real number semantics (for elementary operations) instead of floating-point semantics. This exercises our ability to correctly link with RealLib. It also provides us with an almost complete foundation of LLVM infrastructure for continuation with the multiple week project.

8. Collaboration
We have divided our one week project into two major pieces:

- get the RealLib library working and generate a simple test suite
- modify the LLVM interpreter to use real number semantics

We've decided that Dan and Paola will work on the first bullet while Yihe and Hannah concurrently work on the second bullet.

Our ideal, final project consists of a number of pieces:

- modifications to the LLVM interpreter to call RealLib
- generating a set test of test cases to debug our interpreter
- gathering a corpus of floating-point-arithmetic-using programs
- running experiments on the full corpus
- writing the report, generating figures, computing statistics, and summarizing background knowledge

Acknowledgments
We acknowledge Eddie's face

References