15-745: Optimizing Compilers for Modern Architectures

Project Proposal

Sol Boucher and Goran Žužić

March 18, 2016

1 Introduction

Logical bugs are a big source of programming errors and can be notoriously hard to catch. While there is no hope to detect all logical mistakes in source code, one might hope to design bug detection software to capture a specific type of probable bugs. To this end, we propose a static bug detection technique that uses dimensional analysis to flag possible programmer mistakes.

Examine the C++ source code in Listing 1. If we remove the statement "/*sq_dist*(...)" marked in red, the code contains a dimensional bug. To see the bug, it is important to know that the variables `v, g` and `pts[.]` hold 2D points. Let's denote the dimension of the unit distance of this 2D plane as E. It can be inferred that the dimension of `dot_prod` is E^2. Without the statement in red, the dimension of `alpha` is also E^2. But this implies that `pts[i].x + alpha*v.x` will be adding together variables of dimension E and E^3, a clear dimensional bug. The proposed method vows to catch such bugs.

Listing 1: Example of a dimensional bug

```
double dot_prod = (g.x-pts[i].x)*v.x + (g.y-pts[i].y)*v.y;
double alpha = dot_prod / sq_dist(v, {0, 0});
if (...) {
  Pt cand = {pts[i].x + alpha*v.x, pts[i].y + alpha*v.y};
  ...
}
```

This example from Listing 1 comes from a competitive-programming event (edited for clarity) and this was the actual bug one of the authors made during the contest.

2 Scope and Variations

We propose a static bug detection method that uses dimensional analysis. The method is purely non-parametric and doesn't require any additional annotation from the programmer. While such a dimensional bugs can often be found easily with unit testing, there are certain scenarios where we believe our method can be useful: *i)* in cases where unit-tests don't cover

all code paths or where the unit-test use trivial values for (0 or 1) for dimensional variables *ii)* in the context of competitive programming for fast static bug detection.

The approach is static (compiler-assisted) and is useful for debugging. Therefore we believe it is well within the scope of the 15-745 course.

Before we describe the methodology of the method itself, it is worth mentioning that such a dimensional analysis is a fairly novel approach. The closest prior art we were able to find are ???? [1].

TODO: find prior art

Below, we provide our goals for this project. If the problem is more challenging than we expect, we will only complete the 75% goals; if the project is roughly as challenging as we expect, then we will complete some subset of the 100% goals; and, if we make progress at a quicker rate than anticipated, we will consider some of the 125% goals.

75% goal If the dimensional analysis proves too hard or produces too many false-positives to be useful we will consider a reduced-scope problem of "mod-consistency-detection". Basically, each time a variable stores a value that modulo some P, we are to expect that this variable will always contain some value calculated modulo P. If it appears this is not the case, we issue a warning. This problem can be viewed not only as a semantic bug detection, but also as a dimensional analysis problem if we consider "int modulo P" as a separate type.

100% goal We would like to have a working dimensional analysis tool that will produce useful debugging output when given as input a contest problem with a dimensional analysis bug. Additionally, if the framework appears too-impractical, we might consider adding some programmer-annotation tools to help guide the analysis.

125% goal As our most ambitious goal, we will try to design a practical framework that could be fed mature software codebases and have a reasonable output.

3 Project Timeline

4 Experiments and Evaluation

As a evaluation method we are considering downloading a vast amount of source codes from open competitive programming sites like `codeforces.com` and finding dimensional bugs. Obvisouly, such evaluation will require a lot of manual labor of checking for false-positives and it is not clear what is the probability that a random source code contains a dimensional-analysis bug. However, our hope is to find a nontrivial amount of bugs and present a case-study we use the tool to find them.

5 Outline of the Paper

References

[1] P. N. Hilfinger. An ada package for dimensional analysis. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 10(2):189–203, 1988.