Project 2 Report

June 5, 2014

1 Overview

> Discuss the high level goals of your work, along with any interesting/key findings.

Difficult, because there were no interesting findings. Consider: "huh, LLVM has an API for some of this stuff I guess," or "Wow C++ is painful." I (marco) will be able to make up some stuff for this once I'm done with CSE (which should soon).

2 Interface Design

> Describe interface. Discuss what alternative designs you may have also considered, and explain the tradeoffs that ultimately led to your choice.

Unsure how honest to be here. Tradeoffs ended up being, how many C++ features can we avoid and still have a functioning C++ program? Other than this issue, describing our interface (flowfunction object, latticepoint object, and a blob of code implementing the worklist algorithm that uses runtime polymorphism to use these objects) is pretty straightforward. I (marco) can write this.

3 Analyses

3.1 Constant Propagation

3.1.1 Mathematical Flow Functions

> define your lattice and flow functions in mathematical notation (i.e., in the style used in the lecture notes and on the midterm).

3.1.2 Implementation Considerations

discuss how you actually went about implementing the lattice and flow functions. For instance, some interesting questions are: what data structure(s) did you use, and how do you represent potentially infinite sets? How are input facts passed to your flow functions, and how do output facts get propagated?

3.2 Available Expressions

3.2.1 Mathematical Flow Functions

define your lattice and flow functions in mathematical notation (i.e., in the style used in the lecture notes and on the midterm).

3.2.2 Implementation Considerations

discuss how you actually went about implementing the lattice and flow functions. For instance, some interesting questions are: what data structure(s) did you use, and how do you represent potentially infinite sets? How are input facts passed to your flow functions, and how do output facts get propagated?

3.3 Range Analysis

3.3.1 Mathematical Flow Functions

define your lattice and flow functions in mathematical notation (i.e., in the style used in the lecture notes and on the midterm).

3.3.2 Implementation Considerations

discuss how you actually went about implementing the lattice and flow functions. For instance, some interesting questions are: what data structure(s) did you use, and how do you represent potentially infinite sets? How are input facts passed to your flow functions, and how do output facts get propagated?

3.4 Intra-Procedural Pointer Analysis

3.4.1 Mathematical Flow Functions

> define your lattice and flow functions in mathematical notation (i.e., in the style used in the lecture notes and on the midterm).

3.4.2 Implementation Considerations

> discuss how you actually went about implementing the lattice and flow functions. For instance, some interesting questions are: what data structure(s) did you use, and how do you represent potentially infinite sets? How are input facts passed to your flow functions, and how do output facts get propagated?

4 Testing

> Make sure to explain assumptions you make about the code you analyze. For instance, for pointer analysis, you may have made some assumptions about the aliasing information known about input parameters. Explain those assumptions and why they are reasonable.
>
> Part of this project is to come up with a useful set of benchmarks on which to test and improve your analysis. Discuss why you chose those benchmarks, and what makes them interesting. If your implementation fails on some benchmarks (there's no shame in it!), then explain why and how the analysis might be improved.

4.1 Benchmarks/Assumptions in Common

Because we have a common pool of benchmarks, we can list them here along with common assumptions on code. More specialized discussion of per-analysis benchmark goes below. Here, we can also introduce the Straight Line Program/Branching Program distinction.

4.2 Constant Propagation

4.3 Available Expressions

4.4 Range Analysis

4.5 Intra-Procedural Pointer Analysis

5 Conclusion/Challenges

> As this project is significantly more exploratory than the first, we want you to tell us what you found particularly interesting/challenging/frustrating. What extensions to the project did you attempt? (for instance, did you try combining the results of analyses, or did you try your hand at interprocedural analysis?). What aspects of LLVM made it easy/hard to implement your design? If you could redo your project with what you know now, what changes would you make?

Also unsure how honest to be here. Most of our challenges had to do with counter-intuitive LLVM design and poor documentation, as well as the unpredictability of C++ features.