CS 6241: Compiler Design

Refined Def-Use Analysis Implementation Using Infeasible Paths Information

Chayne Thrash and Mansour Alharthi

April 7, 2018

Instructor: Professor Santosh Pande

1 Introduction

In this project, we first implemented an algorithm that detects both the inter- and intra-procedural infeasible paths using branch correlation. Utilizing this information, we implemented a demand-driven def-use analysis that discards the chains along infeasible paths and therefor give a more accurate data flow information. Both algorithms were introduced in the paper "Refining Data Flow Information Using Infeasible Paths" by Gupta, Soffa, and Bodik. Finally, we improved the introduced infeasible path detection algorithm by accounting for more cases than the ones mentioned in the paper. We show our improved version with test results in this report.

2 Improved Infeasible Paths Detection

We improved the algorithm presented in the paper by adding the ability to handle binary operations with constants to the infeasible path detection algorithm. Specifically, the added operations were addition, subtraction, multiplication, and both signed and unsigned division. This improvement was implemented by adding a stack to the query that kept track of these binary operations. Whenever one of these operations is encountered, both the opcode and the constant involved are placed onto the stack. Whenever the query variable is assigned to a constant, the stack is replayed onto the assigned constant. The resulting value is then used to determine query resolution.

3 Test Results

We tested our inter-procedural Def-Use analysis on two benchmarks. The tables below show comparisons between the optimized codes generated using the refined def-use analysis with infeasible paths information and the regular Def-Use analysis.

Benchmark	Def-Use pairs# without infeasible paths	Def-Use pairs# with infeasible paths	% of Def-Use pairs removed
H.264/MPEG-4 AVC	368	189	48.6
JPEG-2000	368	189	48.6

Table 1: Def-Use pairs comparison

Benchmark	Static size without infeasible paths	Static size with infeasible paths	% of improvement in static size
H.264/MPEG-4 AVC	18240	18168	.39
JPEG-2000	18240	18168	.39

Table 2: Static size comparison

Benchmark	Execution time without infeasible paths	Execution time with infeasible paths	% of improvement in Execution time
H.264/MPEG-4 AVC	0.016s	0.014s	12.5
JPEG-2000	0.006s	0.005s	16.7

Table 3: Execution time comparison

4 Work Breakdown

Intra-procedural Infeasible Paths Detection	Intra-procedural Demand-Driven Def-Use Analysis	Inter-procedural Infeasible Paths Detection	Inter-procedural Demand-Driven Def-Use Analysis	Node
Chayne	Mansour	Chayne	Mansour	Chayne

Table 4: breakdown of work among team members