CS 6241: Compiler Design

Solving Constant Propagation Data-flow Problem Using Detected Destructive Merges Information

Chayne Thrash and Mansour Alharthi

April 22, 2018

Instructor: Professor Santosh Pande

1 Introduction

In this project, we implemented the algorithm introduced in paper "Comprehensive Path-sensitive Data-flow Analysis" by Thakur and Govindarajan. The algorithm first detects the nodes where two or more paths merge; causing constant variables defined in one or more of those paths no longer feasible for propagating. After that, the algorithm look for the influenced nodes; the nodes of which will be optimized in case the destructive merge is eliminated. Finally, the algorithm duplicates the nodes that are in the *region of influence*, and thus eliminating the destructive merge allowing for the constant propagation to the influenced nodes. The algorithm represents a trade-off between code size and data-flow precision. We apply the algorithm on only the two top fittest destructive merges, as per the definition of fitness in the paper.

2 Test Results

We tested our implementation on two benchmarks. We compare our implementation with sparse conditional constant propagation technique by Wegman-Zadeck implemented in LLVM.

Benchmark	Static size with sparse conditional constant propagation	Static size with destructive merge elimination	% of increase in static size
bzip2	88776B	105160B	%18
gzip	77584B	85776B	%11

Table 1: Static size comparison

Benchmark	Constants propagated# with sparse conditional constant propagation	Constants propagated# with destructive merge elimination	% of Constants propagated increased
bzip2	56	107	%91
gzip	31	106	%240

Table 2: Constants propagated comparison

Benchmark	Execution time with sparse conditional constant propagation	Execution time with destructive merge elimination	% of change in Execution time
bzip2	0.357s	0.909s	%154
gzip	0.073s	0.176s	%150

Table 3: Execution time comparison when a compressing 1.2MB file

Figures 1 and 2 show examples of a function CFG with SCCP and Destructive merges transformations. The function is taken from the bzip2 benchmark codebase. This example shows the elimination of the PHI function in the return block by splitting the path resulting in the propagation of the returned values all the way to the exit points.

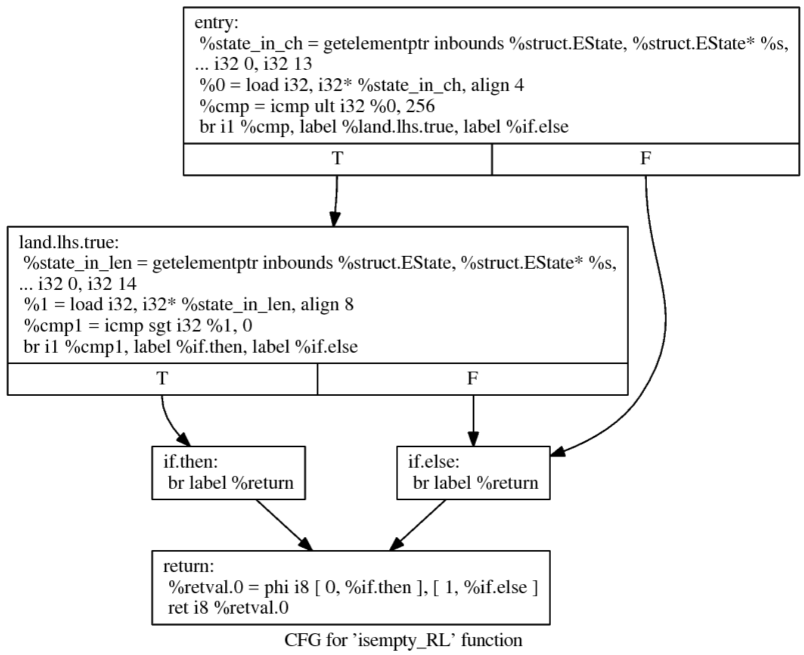

Figure 1: function isempty_RL in bzip2 with SCCP transformation

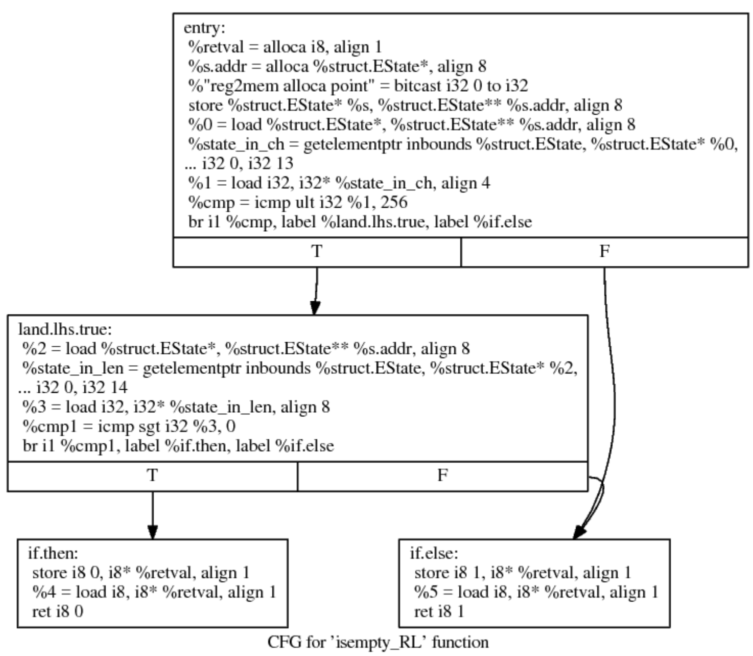

Figure 2: function isempty_RL in bzip2 with Destructive merge transformation

3 Work Breakdown

Destructive Merges Detection	Split Graph and CFG Reconstruction	Preliminary Reachability Analysis	Report	Integrating and testing
Mansour	Chayne	Chayne	Mansour	Chayne&Mansour

Table 4: breakdown of work among team members