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
B2	D	E	F

Table 1: Static size comparison

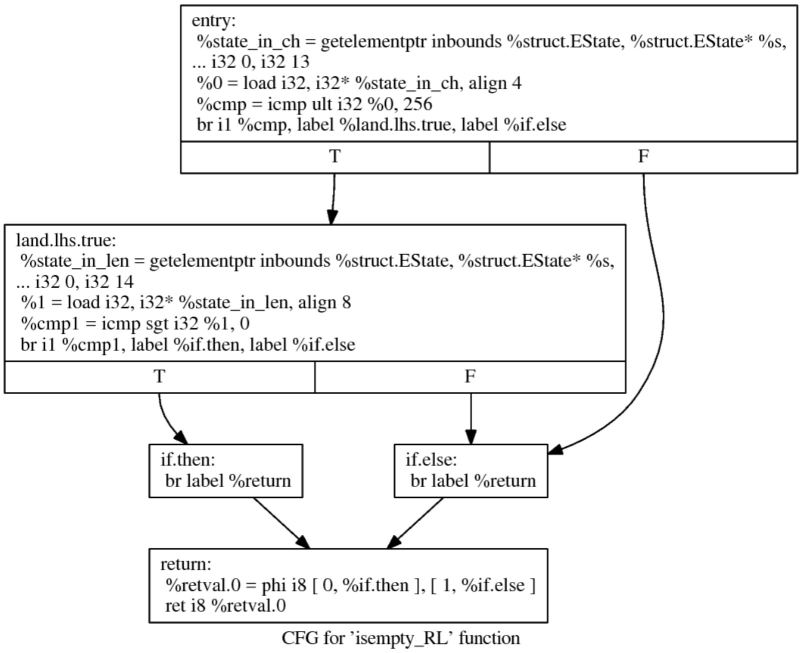
Benchmark	Constants propagated# with sparse conditional constant propagation	Constants propagated# with destructive merge elimination	% of Constants propagated increased
bzip2	56	107	%91
B2	D	E	F

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 (Compressing 1.2MB file)	0.357	0.835s	%133
B2	D	E	F

Table 3: Execution time comparison

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.



```
entry:
 %state_in_ch = getelementptr inbounds %struct.EState, %struct.EState* %s,
... i32 0, i32 13
 %0 = load i32, i32* %state_in_ch, align 4
 %cmp = icmp ult i32 %0, 256
 br i1 %cmp, label %land.lhs.true, label %if.else
```

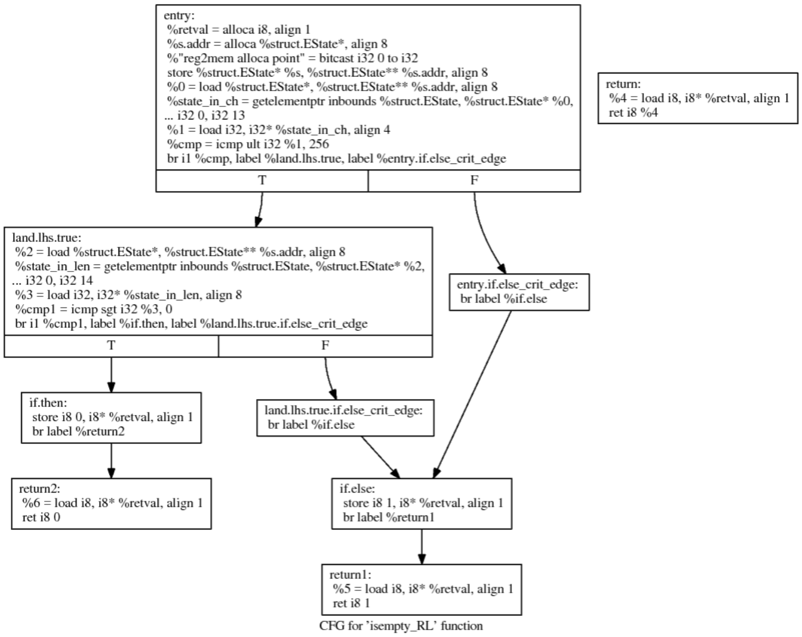
```
land.lhs.true:
 %state_in_len = getelementptr inbounds %struct.EState, %struct.EState* %s,
... i32 0, i32 14
 %1 = load i32, i32* %state_in_len, align 8
 %cmp1 = icmp sgt i32 %1, 0
 br i1 %cmp1, label %if.then, label %if.else
```

```
if.then:
 br label %return
```

```
if.else:
 br label %return
```

```
return:
 %retval.0 = phi i8 [ 0, %if.then ], [ 1, %if.else ]
 ret i8 %retval.0
```

Figure 1: function isempty_RL in bzip2 with SCCP transformation



```
entry:
 %retval = alloca i8, align 1
 %s.addr = alloca %struct.EState*, align 8
 %"reg2mem alloca point" = bitcast i32 0 to i32
 store %struct.EState* %s, %struct.EState** %s.addr, align 8
 %0 = load %struct.EState*, %struct.EState** %s.addr, align 8
 %state_in_ch = getelementptr inbounds %struct.EState, %struct.EState* %0,
... i32 0, i32 13
 %1 = load i32, i32* %state_in_ch, align 4
 %cmp = icmp ult i32 %1, 256
 br i1 %cmp, label %land.lhs.true, label %entry.if.else_crit_edge
```

```
return:
 %4 = load i8, i8* %retval, align 1
 ret i8 %4
```

```
land.lhs.true:
 %2 = load %struct.EState*, %struct.EState** %s.addr, align 8
 %state_in_len = getelementptr inbounds %struct.EState, %struct.EState* %2,
... i32 0, i32 14
 %3 = load i32, i32* %state_in_len, align 8
 %cmp1 = icmp sgt i32 %3, 0
 br i1 %cmp1, label %if.then, label %land.lhs.true.if.else_crit_edge
```

```
entry.if.else_crit_edge:
 br label %if.else
```

```
if.then:
 store i8 0, i8* %retval, align 1
 br label %return2
```

```
land.lhs.true.if.else_crit_edge:
 br label %if.else
```

```
return2:
 %6 = load i8, i8* %retval, align 1
 ret i8 0
```

```
if.else:
 store i8 1, i8* %retval, align 1
 br label %return1
```

```
return1:
 %5 = load i8, i8* %retval, align 1
 ret i8 1
```

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