Exploiting and improving LLVM's data flow analysis using a superoptimizer

Jubi Taneja

John Regehr, Raimondas Sasnauskas, Peter Collingbourne, Yang Chen, Jeroen Ketema

University of Utah

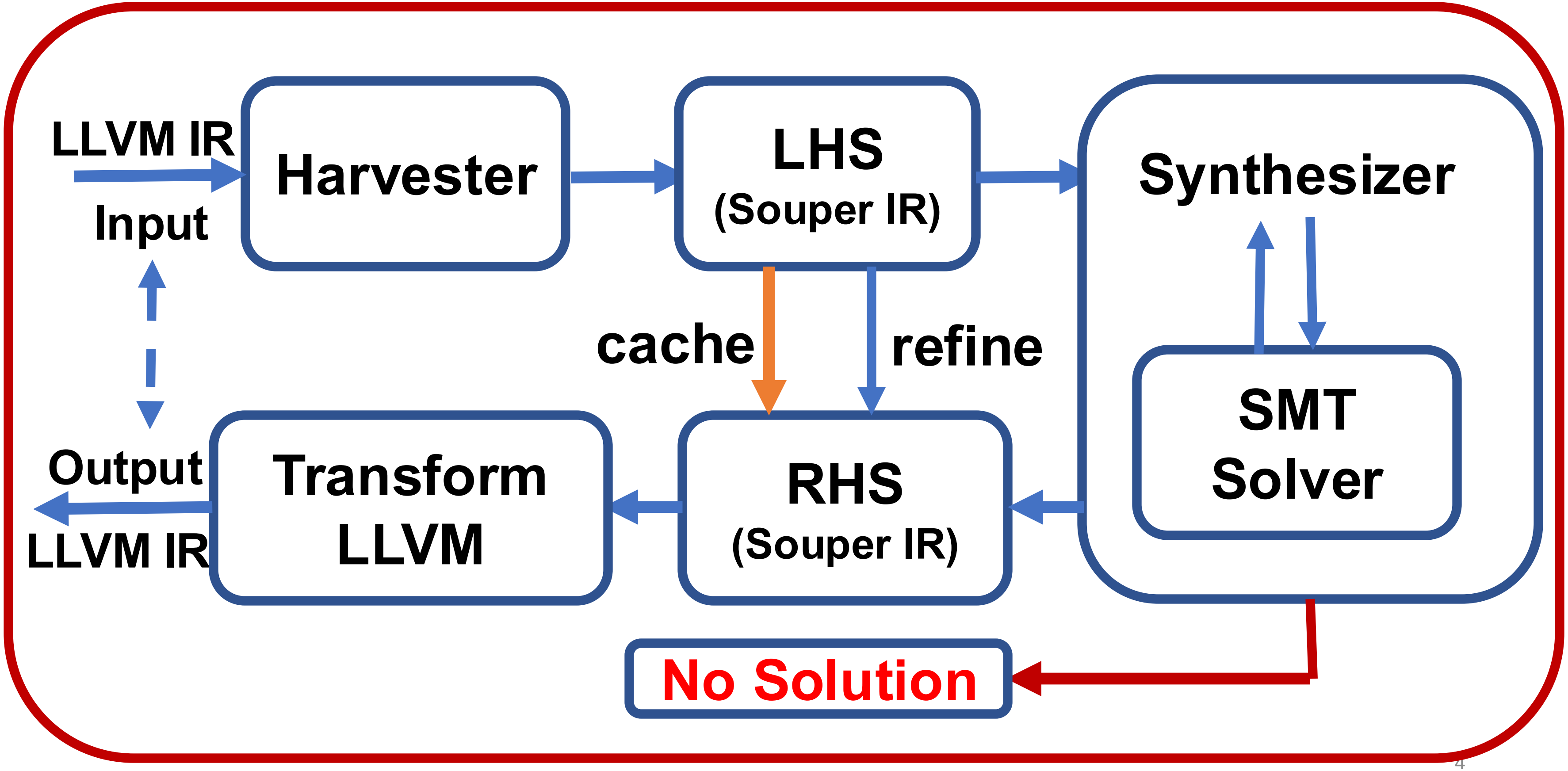
- **Goal: Automatically discover peephole optimizations**

- **We created a synthesis based superoptimizer: Souper**

```
define i32 @foo(i32 %x1) {
  %0 = and 0x55555555, %x1
  %1 = lshr i32 %x1, 1
  %2 = and 0x55555555, %1
  %3 = add i32 %0, %2
  %4 = and 0x33333333, %3
  %5 = lshr i32 %3, 2
  %6 = and 0x33333333, %5
  %7 = add i32 %4, %6
  %8 = and 0x0F0F0F0F, %7
  %9 = lshr i32 %7, 4
  %10 = and 0x0F0F0F0F, %9
  %11 = add i32 %8, %10
  %12 = and 0x00FF00FF, %11
  %13 = lshr i32 %11, 8
  %14 = and 0x00FF00FF, %13
  %15 = add i32 %12, %14
  %16 = and 0x0000FFFF, %15
  %17 = lshr i32 %15, 16
  %18 = and 0x0000FFFF, %17
  %19 = add i32 %16, %18
  ret i32 %19
}
```

```
define i32 @foo(i32 %x1) {
foo0:
  %0 = call i32 @llvm.ctpop.i32(i32 %x1)
  ret i32 %0
}
```

libSouperPass.so



LLVM IR

Input

Harvester

LHS

(Souper IR)

Synthesizer

cache

refine

SMT

Solver

Output

LLVM IR

Transform

LLVM

RHS

(Souper IR)

No Solution

- **Souper makes clang-5.0 text segment 1.6 MB smaller in a Release+Assertions build**
- **~10 patches in LLVM mention Souper**

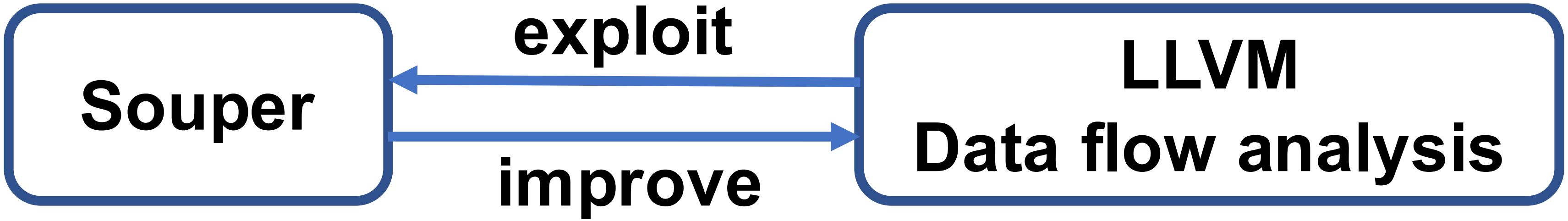
Integrating Souper with data flow analysis

```
define i32 @foo() {
  ...
  // isKnownToBeAPowerOfTwo(%x) == true
  %2 = call i32 @llvm.ctpop.i32(i32 %x)
  ret i32 %2
}
```

```
ret i32 1
```

Souper exploits LLVM’s data flow analyses

- Power of Two
- Known bits
- Non-negative
- Negative
- Number of sign bits
- Demanded bits



Souper

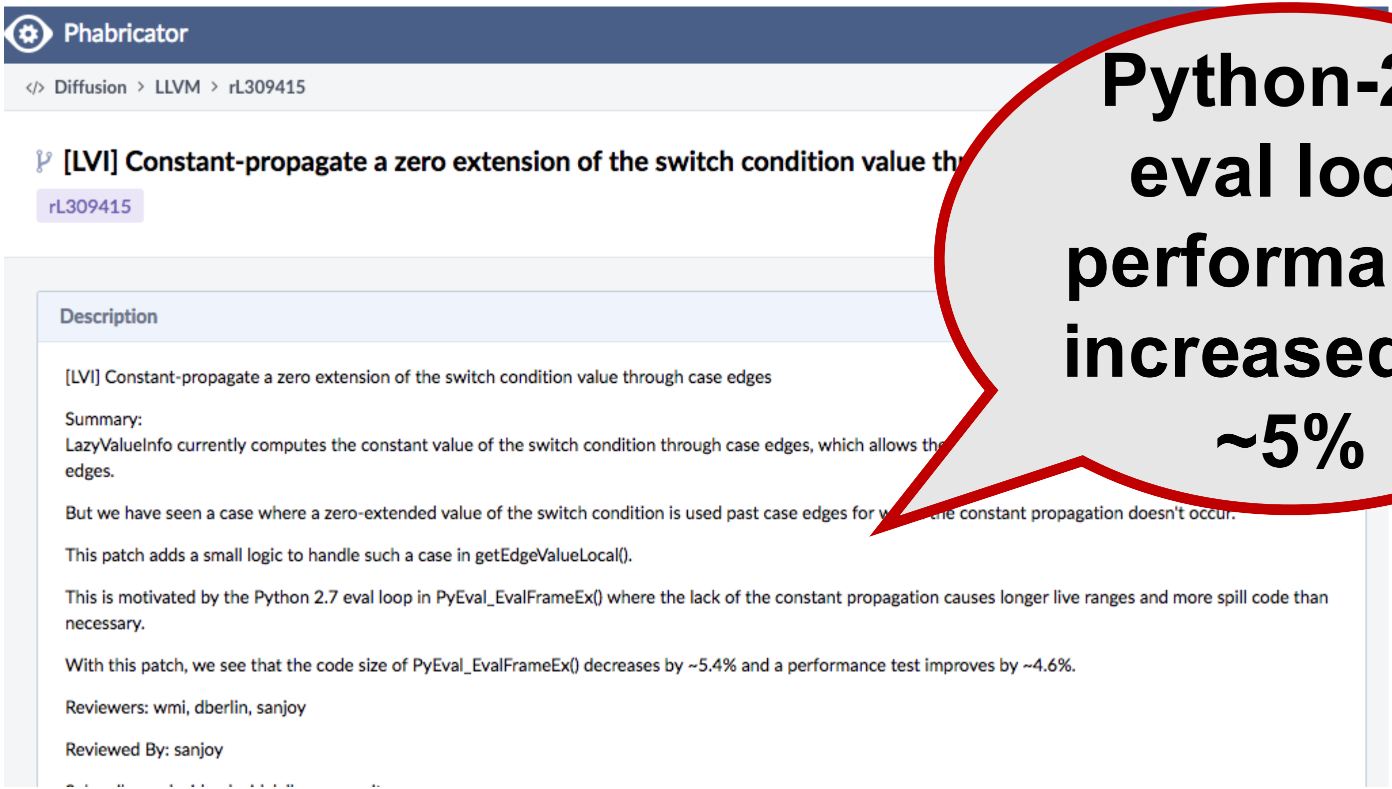
exploit

improve

LLVM

Data flow analysis

An imprecision in Lazy Value Info



[LVI] Constant-propagate a zero extension of the switch condition value thr

rL309415

Description

[LVI] Constant-propagate a zero extension of the switch condition value through case edges

Summary:
LazyValueInfo currently computes the constant value of the switch condition through case edges, which allows the
edges.

But we have seen a case where a zero-extended value of the switch condition is used past case edges for w the constant propagation doesn't occur.

This patch adds a small logic to handle such a case in getEdgeValueLocal().

This is motivated by the Python 2.7 eval loop in PyEval_EvalFrameEx() where the lack of the constant propagation causes longer live ranges and more spill code than necessary.

With this patch, we see that the code size of PyEval_EvalFrameEx() decreases by ~5.4% and a performance test improves by ~4.6%.

Reviewers: wmi, dberlin, sanjoy

Reviewed By: sanjoy

Python-2.7 eval loop performance increased by ~5%

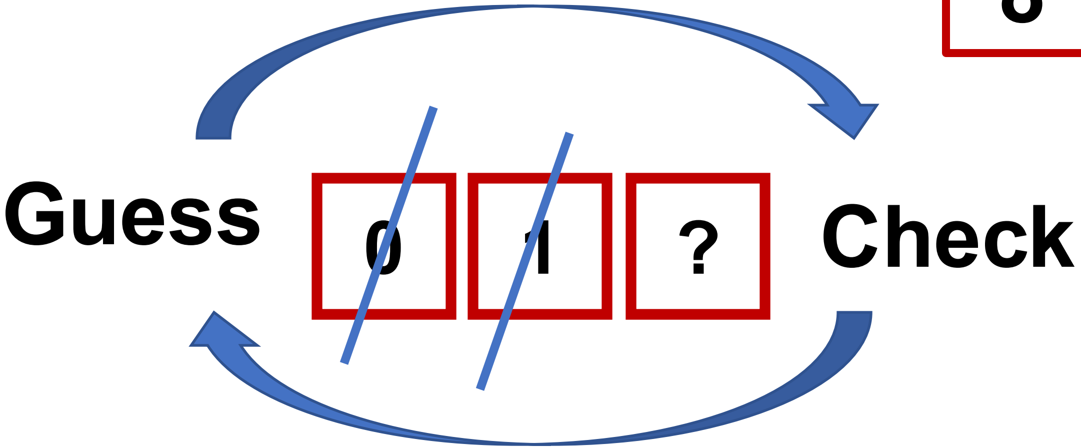
Imprecision in computeKnownBits

```
define i16 @foo(i16 %x) {
  …
  %0 = shl i16 8, %x
  …
}
```

LLVM: ????????????????

Heuristic technique to compute near optimal data flow facts derived by Souper

8 << x



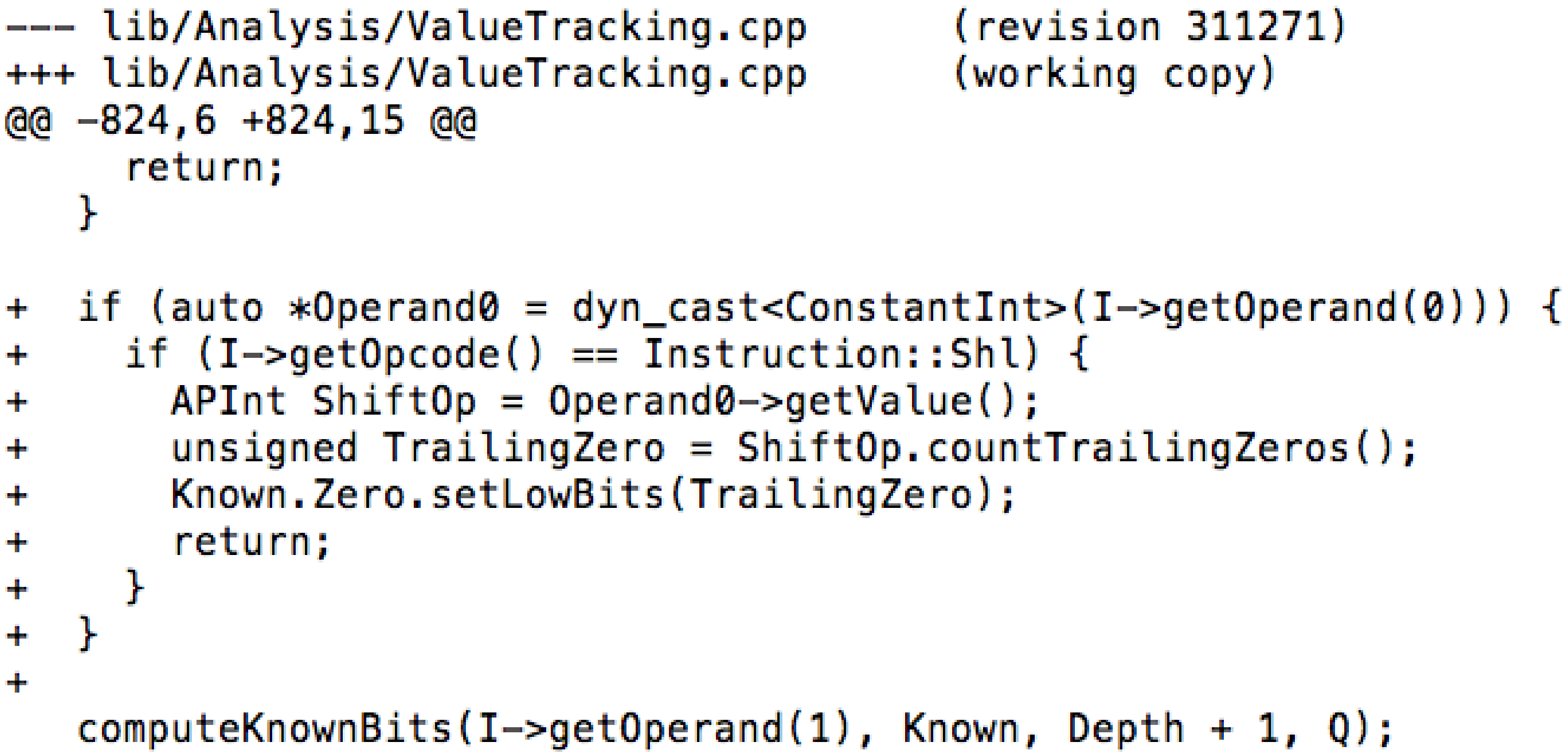
Souper: ?   ????????????   000

LLVM: ????????????????

```
define i16 @foo(i16 %x) {
  ...
  %0 = shl i16 8, %x
  ...
}
```

LLVM: ????????????????

Souper: ?????????????000



```
--- lib/Analysis/ValueTracking.cpp      (revision 311271)
+++ lib/Analysis/ValueTracking.cpp      (working copy)
@@ -824,6 +824,15 @@
     return;
   }

+  if (auto *Operand0 = dyn_cast<ConstantInt>(I->getOperand(0))) {
+    if (I->getOpcode() == Instruction::Shl) {
+      APInt ShiftOp = Operand0->getValue();
+      unsigned TrailingZero = ShiftOp.countTrailingZeros();
+      Known.Zero.setLowBits(TrailingZero);
+      return;
+    }
+  }
+
   computeKnownBits(I->getOperand(1), Known, Depth + 1, Q);
```

Summary

Souper is a peephole superoptimizer that can both improve and exploit LLVM’s data flow analysis.

Souper is open source: https://github.com/google/souper