Alive: Provably Correct InstCombine Optimizations

David Menendez
Santosh Nagarakatte
Rutgers University

John Regehr
University of Utah

Nuno Lopes
Microsoft Research
Can We Trust Compilers?

- Any large software project will have bugs
- LLVM is no exception
  - CSmith project found 203 bugs by random testing
- InstCombine is especially buggy
I noticed that one of the reassociate regression tests -- 2002-07-09-DominanceProblem -- was failing. instcombine was transforming

```c
%A.neg = sub int 0, %A
%neg = sub int 0, 1
%X = add int %.neg, 1
%Y.neg.ra = add int %A, %X
%r = add int %A.neg, %Y.neg.ra
ret int %r

into

%Y.neg.ra = add int %A, 0
%r = add int %A.neg, %Y.neg.ra
ret int %r
```

without reexamining %Y.neg.ra to see that it is clearly replacable by %A. The problem is that instcombine's runOnFunction does not add uses of a constant-folded instruction into its worklist for examination.
[LLVMbugs] [Bug 1976] New: instcombine doesn't fold x*y+x*z to x*(y+z)

bugzilla-daemon at cs.uiuc.edu  bugzilla-daemon at cs.uiuc.edu
Sat Feb 2 22:41:40 CST 2008

- Previous message: [LLVMbugs] [Bug 1975] dag isel emitter isels wrong flag result
- Next message: [LLVMbugs] [Bug 1976] instcombine doesn't fold x*y+x*z to x*(y+z)
- Messages sorted by: [ date ] [ thread ] [ subject ] [ author ]

http://llvm.org/bugs/show_bug.cgi?id=1976

    Summary: instcombine doesn't fold x*y+x*z to x*(y+z)
    Product: libraries
    Version: trunk
    Platform: All
    OS/Version: All
    Status: NEW
    Severity: enhancement
    Priority: P2
    Component: Scalar Optimizations
    AssignedTo: nicholas at mxc.ca
    ReportedBy: nicholas at mxc.ca
    CC: llvmbugs at cs.uiuc.edu

Created an attachment (id=1369)
--> (http://llvm.org/bugs/attachment.cgi?id=1369)
proposed patch (a little ugly)

Instcombine should be able to fold:

define i8 @test1(i8 %x, i8 %y, i8 %z) {
    %A = mul i8 %x, %y
    %B = mul i8 %x, %z
    %C = add i8 %A, %B
    ret i8 %C
}
**[LLVMbugs] [Bug 1066] NEW: Assertion failed in InstCombine**

*bugzilla-daemon at cs.uiuc.edu*  [bugzilla-daemon at cs.uiuc.edu](mailto:bugzilla-daemon at cs.uiuc.edu)

*Sat Dec 23 18:07:40 CST 2006*

- Previous message:  [LLVMbugs] [Bug 1065] Crash in InstCombine
- Next message:  [LLVMbugs] [Bug 1066] Assertion failed in InstCombine
- Messages sorted by:  [date]  [thread]  [subject]  [author]

[llvm.org/bugs/show_bug.cgi?id=1066](http://llvm.org/bugs/show_bug.cgi?id=1066)

Summary:  Assertion failed in InstCombine
Product:  libraries
Version:  trunk
Platform:  PC
OS/Version:  Linux
Status:  NEW
Severity:  normal
Priority:  P2
Component:  Scalar Optimizations
AssignedTo:  unassignedbugs at nondot.org
ReportedBy:  asl at math.spbu.ru

```
$ ./opt -instcombine bugpoint-reduced-simplified.bc -o test_bc.bc -f
llvm::Value::replaceAllUsesWith(llvm::Value*): Assertion "New->getType() ==
getType() & & "replaceAllUses of value with new value of different type!"
'' failed.
./opt((anonymous namespace)::PrintStackTrace()+0x1f)[0x836a7ff]
/lib/libc.so.6(abort+0xbeb)[0xb7d81133]
/lib/libc.so.6(__assert_fail+0xbeb)[0xb7d794f3]
./opt[0x832f26c]
```

Bytecode attached
[LLVMbugs] [Bug 2295] New: incorrect optimization (instcombine)

bugzilla-daemon at cs.uiuc.edu bugzilla-daemon at cs.uiuc.edu
Wed May 7 17:41:31 CDT 2008

- Previous message: [LLVMbugs] [Bug 2025] llvm-gcc4.2 won't build if using LLVM release build and bootstrap enabled
- Next message: [LLVMbugs] [Bug 2295] incorrect optimization (instcombine)
- Messages sorted by: [ date ] [ thread ] [ subject ] [ author ]

http://llvm.org/bugs/show_bug.cgi?id=2295

Summary: incorrect optimization (instcombine)
Product: new-bugs
Version: unspecified
Platform: PC
OS/Version: Windows NT
Status: NEW
Severity: normal
Priority: P2
Component: new bugs
AssignedTo: unassignedbugs_at_nondot.org
ReportedBy: scott.llvm_at_h4ck3r.net
CC: llvmbugs_at_cs.uiuc.edu

Created an attachment (id=1621)
   --> (http://llvm.org/bugs/attachment.cgi?id=1621)
   .ll to demonstrate mis-optimization

Using LLVM 2.2:

llvm-as -f x.ll
opt -instcombine -f x.bc -o x.opt.bc
lli x.bc
lli x.opt.bc

The first outputs (as expected):
90 (2)
and the second (optimized) incorrectly outputs:
33 (1)

I was unable to get bugpoint to work, but the attached file has been manually minimized (73 lines). Happens with "opt -std-compile-opts" also, but "opt
-instcombine" is all that's necessary.

It appears two different pointers into allocated memory are being conflated.
[LLVMbugs] [Bug 3953] New: Instcombine quadratic in add chain length

bugzilla-daemon at cs.uiuc.edu  bugzilla-daemon at cs.uiuc.edu
Mon Apr 6 16:10:20 CDT 2009

- Previous message: [LLVMbugs] [Bug 3678] Invalid input/output constraint when compiling FreeBSD's libm/ OpenSSL on AMD64
- Messages sorted by: [date] [thread] [subject] [author]

http://llvm.org/bugs/show_bug.cgi?id=3953

Summary: Instcombine quadratic in add chain length
Product: libraries
Version: trunk
Platform: PC
OS/Version: All
Status: NEW
Severity: normal
Priority: P2
Component: Scalar Optimizations
AssignedTo: unassignedbugs_at_nondot.org
ReportedBy: iyasskin at google.com
CC: llvmbugs at cs.uiuc.edu, nlewycky at google.com

Created an attachment (id=2799)
--> (http://llvm.org/bugs/attachment.cgi?id=2799)
Generator for add chains that make instcombine quadratic

The attached script generates .ll files of the form:

$ ./instcombine_test3.py 3
declare void @use(i32)

define void @foo(i32 %B0) {
   %A1 = add i32 %B0, 1
   %B1 = add i32 %A1, -1
   call void @use(i32 %B1)
The following transformation is incorrect: 

```c
fsub(a, fsub(b, c)) =>
fadd(a, fsub(c, b))
```

For example:

```c
c
float
func(float a, float b) {
    return -(a - b);
}
```

`llvm-gcc` generates:

```c
define float @func(float %a, float %b) nounwind {
  entry:
    %tmp3 = sub float %a, %b ; <float> [#uses=1]
    %tmp4 = sub float -0.000000e+00, %tmp3 ; <float> [#uses=1]
  ret float %tmp4
}
```
[LLVMbugs] [Bug 4908] New: infinite loop in instcombine

bugzilla-daemon at cs.uiuc.edu  bugzilla-daemon at cs.uiuc.edu
Sat Sep 5 21:18:40 CDT 2009

- Previous message: [LLVMbugs] [Bug 4907] llvm fails to build, error: 'WeakVH' was not declared
- Next message: [LLVMbugs] [Bug 4908] infinite loop in instcombine
- Messages sorted by: [ date ] [ thread ] [ subject ] [ author ]

http://llvm.org/bugs/show_bug.cgi?id=4908

Summary: infinite loop in instcombine
Product: libraries
Version: trunk
Platform: PC
OS/Version: All
Status: NEW
Severity: normal
Priority: P2
Component: Scalar Optimizations
AssignedTo: unassignedbugs_at_nondot.org
ReportedBy: daniel_at_zuster.org
CC: llvmbugs_at_cs.uiuc.edu

Instcombine loops on:
```c
--
; ModuleID = '<stdin>'
target triple = "x86_64-apple-darwin10.0.0"

define void @short2_int_swap(<1 x i16>*) nocapture %b, i32* nocapture %c)
nounwind ssp {
entry:
  %arrayidx = getelementptr inbounds <1 x i16>* %b, i64 undef ; <<1 x i16>*
  #[uses=1]
    %tmp2 = load <1 x i16>* %arrayidx ; <<1 x i16>>
  %tmp6 = bitcast <1 x i16> %tmp2 to i16 ; <i16> [uses=1]
  %tmp7 = zext i16 %tmp6 to i32 ; <i32> [uses=1]
  %ins = or i32 0, %tmp7 ; <i32> [uses=1]
```
Summary: instcombine deoptimizes testcase
Product: libraries
Version: trunk
Platform: PC
OS/Version: All
Status: NEW
Severity: normal
Priority: P
Component: Scalar Optimizations
Assignee: unassignedbugs at nondot.org
ReportedBy: rafael.espindola at gmail.com
CC: llvmbugs at cs.uiuc.edu

In the attached testcase, instcombine converts

%tmp2 = ptrtoint %struct.Shape* %tmp9 to i64
%and = and i64 %tmp2, -2
%tmp5 = inttoptr i64 %and to %struct.Shape*
%toobool = icmp ne %struct.Shape* %tmp5, null
br il %toobool, label %land.lhs.true, label %if.end

land.lhs.true: ; preds = %entry
  %propid = getelementptr inbounds %struct.Shape* %tmp5, i32 0, i32 2
to
  %toobool = icmp ugt %struct.Shape* %tmp9, inttoptr (i64 1 to %struct.Shape*)
br il %toobool, label %land.lhs.true, label %if.end

land.lhs.true: ; preds = %entry
[LLVMbugs] [Bug 14893] New: Instcombine miscompiles bool, aka i8 !range[0, 2]

bugzilla-daemon at llvm.org bugzilla-daemon at llvm.org
Thu Jan 10 07:52:53 CST 2013

- Previous message: [LLVMbugs] [Bug 14745] Formatting empty function bodies
- Next message: [LLVMbugs] [Bug 14894] New: c++-analyzer does not define __SSE3__
- Messages sorted by: [date] [thread] [subject] [author]

http://llvm.org/bugs/show_bug.cgi?id=14893

Bug #: 14893
Summary: Instcombine miscompiles bool, aka i8 !range[0,2]
Product: libraries
Version: trunk
Platform: All
OS/Version: All
Status: NEW
Severity: normal
Priority: P
Component: Scalar Optimizations
AssignedTo: unassignedbugs_at_nondot.org
ReportedBy: geek4civic_at_gmail.com
CC: baldrick_at_free.fr, llvmbugs_at_cs.uiuc.edu
Classification: Unclassified

target triple = "x86_64-redhat-linux"

define zeroext i8 @_Z3BARv() unnamed_addr nounwind uwtable {
entry:
  %r = alloca i8, align 1
  %tmp = call zeroext i8 @_Z3FOORb(i8* %r) nounwind
  %tmp9 = and i8 %tmp, 1
  %tmp1 = icmp eq i8 %tmp9, 0
  br i1 %tmp1, label "%5", label "%3"
"3"
  %tmp3 = load i8* %r, align 1, !range !0
  %tmp5 = zeroext i8 %tmp3
  %tmp6 = icmp eq i8 %tmp5, 0
  br i1 %tmp6, label "%4", label "%3"
"4"
  br i1 %tmp9, label "%5", label "%3"
"5"
  %tmp2 = load i8* %r, align 1, !range 0
  %tmp7 = zeroext i8 %tmp2
  %tmp8 = icmp eq i8 %tmp7, 0
  br i1 %tmp8, label "%6", label "%3"
"6"
  br i1 %tmp9, label "%5", label "%3"
[LLVMbugs] [Bug 16244] New: instcombine breaks this file

bugzilla-daemon at llvm.org bugzilla-daemon at llvm.org
Thu Jun 6 11:51:35 CDT 2013

- Previous message: [LLVMbugs] [Bug 15069] - Wassign enum assertion with\attribute\((\text{packed})\) enum
- Next message: [LLVMbugs] [Bug 16244] instcombine breaks this file
- Messages sorted by: [date] [thread] [subject] [author]

http://llvm.org/bugs/show_bug.cgi?id=16244

- Bug ID: 16244
- Summary: instcombine breaks this file
- Product: libraries
- Version: trunk
- Hardware: PC
- OS: Linux
- Status: NEW
- Severity: normal
- Priority: P
- Component: Scalar Optimizations
- Assignee: unassignedbugs at nondot.org
- Reporter: rafael.espindola at gmail.com
- CC: llvmbugs at cs.uiuc.edu
- Classification: Unclassified

Created attachment 10637

--> http://llvm.org/bugs/attachment.cgi?id=10637&action=edit
testcase

This is a reduction from HashCombineRangeBasicTest failing when building with
dragonegg.

This is a recent regression. I am bisecting it.

bugzilla-daemon@llvm.org bugzilla-daemon@llvm.org
Fri Aug 2 09:56:05 CDT 2013

- Previous message: [LLVMbugs] [Bug 16450] Combining CHECK-DAG and CHECK-NOT sometimes causes an incorrect error-less failure
- Next message: [LLVMbugs] [Bug 16777] New: Clang crash caused by SLP vectorizer
- Messages sorted by: [ date ] [ thread ] [ subject ] [ author ]

http://llvm.org/bugs/show_bug.cgi?id=16776

Bug ID: 16776
Summary: Instcombine transformation causes poor vector codegen
[SSE4]
Product: new-bugs
Version: trunk
Hardware: PC
OS: All
Status: NEW
Severity: normal
Priority: P
Component: new bugs
Assignee: unassignedbugs@nondot.org
Reporter: matt_at_pharr.org
CC: llvmbugs@cs.uiuc.edu
Classification: Unclassified

Created attachment 10973
  --> http://llvm.org/bugs/attachment.cgi?id=10973&action=edit

test case

The attached test case does a vector compare of a <16 x i8> value with zero and then a vector select based on the comparison to negate elements that are less than zero (i.e. computes the absolute value). If I run it through llc as is, a single glorious PABSB instruction is generated:

    pabsb %xmm0, %xmm0

However, if I run "opt -instcombine bug2.ll | llc -o -", I get a 13 instruction sequence instead of the PABSB:
[LLVMbugs] [Bug 18600] New: [InstCombine] assert "Value::replaceAllUsesWith(<null>) is invalid!"

bugzilla-daemon at llvm.org  bugzilla-daemon at llvm.org  
Fri Jan 24 08:09:35 CST 2014

- Previous message: [LLVMbugs] [Bug 18599] New: AsmParser::parseDirectiveMacro can't parse recursive macro definition
- Next message: [LLVMbugs] [Bug 18600] [InstCombine] assert "Value::replaceAllUsesWith(<null>) is invalid!"
- Messages sorted by: [ date ] [ thread ] [ subject ] [ author ]

http://llvm.org/bugs/show_bug.cgi?id=18600

Bug ID: 18600
Summary: [InstCombine] assert "Value::replaceAllUsesWith(<null>) is invalid!"
Product: libraries
Version: 3.3
Hardware: PC
OS: Linux
Status: NEW
Severity: normal
Priority: P
Component: Scalar Optimizations
Assignee: unassignedbugs_at_nondot.org
Reporter: jfonseca_at_vmware.com
CC: llvmbugs_at_cs.uiuc.edu
Classification: Unclassified

Created attachment 11930
-->
http://llvm.org/bugs/attachment.cgi?id=11930&action=edit

bugpoint-reduced-simplified.ll

$ gdb --args ~/work/vmware/llvm/llvm/build/linux-x86_64/Debug+Asserts/bin/opt
bugpoint-reduced-simplified.ll -instcombine -disable-output
(gdb) r
Starting program:
/home/jfonseca/work/vmware/llvm/llvm/build/linux-x86_64/Debug+Asserts/bin/opt
bugpoint-reduced-simplified.ll -instcombine -disable-output
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
opt: /home/jfonseca/work/vmware/llvm/llvm/lib/IR/Value.cpp:304: void llvm::Value::replaceAllUsesWith(llvm::Value*): Assertion `New &&
"Value::replaceAllUsesWith(<null>) is invalid!"' failed.
[LLVMbugs] [Bug 18745] New: Assertion in InstCombine: "getOperand() out of range!"

bugzilla-daemon at llvm.org  bugzilla-daemon at llvm.org
Wed Feb 5 14:15:19 CST 2014

- Previous message: [LLVMbugs] [Bug 18744] New: Clang crashes when instantiating a function template which uses decltype
- Next message: [LLVMbugs] [Bug 18745] Assertion in InstCombine: "getOperand() out of range!"
- Messages sorted by: [ date ] [ thread ] [ subject ] [ author ]

http://llvm.org/bugs/show_bug.cgi?id=18745

Bug ID: 18745
Summary: Assertion in InstCombine: "getOperand() out of range!"
Product: libraries
Version: trunk
Hardware: PC
OS: All
Status: NEW
Severity: normal
Priority: P
Component: Scalar Optimizations
Assignee: unassignedbugs_at_nondot.org
Reporter: jordan_rose_at_apple.com
CC: llvmbugs_at_cs.uiuc.edu
Classification: Unclassified

Compiling the attached file leads to an assertion failure in InstCombine. This started somewhere between r200568 and r200645. (Unfortunately large range on this particular buildbot.)

Assertion failed: (i_noCapture < OperandTraits<PHINode>::operands(this) && "getOperand() out of range!"), function getOperand, file /Volumes/Lore/llvm-public/llvm/include/llvm/IR/Instructions.h, line 2178.

---

clang -ccc -triple x86_64-apple-macosx10.9.0 -emit-obj -disable-free
-main-file-name bufpage.c -mrelocation-model pic -pic-level 2 -mdisable-fp-elim
-relaxed-aliasing -masm-verbose -munwind-tables -target-cpu core2
-target-linker-version 142 -O2 -Wall -Wmissing-prototypes -Wpointer-arith
-Wdeclaration-after-statement -Wendif-labels -Wformat-security -ferror-limit 19
-fmessage-length 0 -fwrapv -stack-protector 1 -mstackrealign -fblocks
Why Is InstCombine Buggy?

- It's huge:
  - over 20,000 lines of code
  - `visitICmpInst` alone is 924 lines
- Complicated to write
- LLVM Semantics are subtle
- Hard to tell when the code is correct
For example...
{  
  Value *Op1C = Op1;
  BinaryOperator *BO = dyn_cast<BinaryOperator>(Op0);
  if (!BO ||
      (BO->getOpcode() != Instruction::UDiv &&
       BO->getOpcode() != Instruction::SDiv)) {
    Op1C = Op0;
    BO = dyn_cast<BinaryOperator>(Op1);
  }
  Value *Neg = dyn_castNegVal(Op1C);
  if (BO && BO->hasOneUse() &&
      (BO->getOperand(1) == Op1C || BO->getOperand(1) == Neg) &&
      (BO->getOpcode() == Instruction::UDiv ||
       BO->getOpcode() == Instruction::SDiv)) {
    Value *Op0BO = BO->getOperand(0), *Op1BO = BO->getOperand(1);

    // If the division is exact, X % Y is zero, so we end up with X or -X.
    if (PossiblyExactOperator *SDiv = dyn_cast<PossiblyExactOperator>(BO))
      if (SDiv->isExact()) {
        if (Op1BO == Op1C)
          return ReplaceInstUsesWith(I, Op0BO);
        return BinaryOperator::CreateNeg(Op0BO);
      }

    Value *Rem;
    if (BO->getOpcode() == Instruction::UDiv)
      Rem = Builder->CreateURem(Op0BO, Op1BO);
    else
      Rem = Builder->CreateSRem(Op0BO, Op1BO);
    Rem->takeName(BO);
    if (Op1BO == Op1C)
      return BinaryOperator::CreateSub(Op0BO, Rem);
    return BinaryOperator::CreateSub(Rem, Op0BO);
  }
}
// (X / Y) * Y = X - (X \% Y)
// (X / Y) * -Y = (X \% Y) - X
{
  Value *Op1C = Op1;
  BinaryOperator *BO = dyn_cast<BinaryOperator>(Op0);
  if (!BO ||
      (BO->getOpcode() != Instruction::UDiv &&
       BO->getOpcode() != Instruction::SDiv)) {
    Op1C = Op0;
    BO = dyn_cast<BinaryOperator>(Op1);
  }
  Value *Neg = dyn_castNegVal(Op1C);
  if (BO && BO->hasOneUse() &&
      (BO->getOperand(1) == Op1C || BO->getOperand(1) == Neg) &&
      (BO->getOpcode() == Instruction::UDiv ||
       BO->getOpcode() == Instruction::SDiv)) {
    Value *Op0BO = BO->getOperand(0), *Op1BO = BO->getOperand(1);
    // If the division is exact, X \% Y is zero, so we end up with X or -X.
    if (PossiblyExactOperator *SDiv = dyn_cast<PossiblyExactOperator>(BO))
      if (SDiv->isExact()) {
        if (Op1BO == Op1C)
          return ReplaceInstUsesWith(I, Op0BO);
        return BinaryOperator::CreateNeg(Op0BO);
      }
    Value *Rem;
    if (BO->getOpcode() == Instruction::UDiv)
      Rem = Builder->CreateURem(Op0BO, Op1BO);
    else
      Rem = Builder->CreateSRem(Op0BO, Op1BO);
    Rem->takeName(BO);
    if (Op1BO == Op1C)
      return BinaryOperator::CreateSub(Op0BO, Rem);
    return BinaryOperator::CreateSub(Rem, Op0BO);
  }
}
Flags can be confusing…
Is This Valid?

Seemingly just

$$(A \times B) \times C = A \times (B \times C)$$
Is This Valid?

%A = -1, %B = 4, %C = 32

%L = mul nsw i8 %A, %B  
%I = mul nsw i8 %L, %C  
%R = mul nsw i8 %B, %C  
%I = mul nsw i8 %A, %R

%L = -4  
%I = -128
Is This Valid?

%A = -1, %B = 4, %C = 32

%L = mul nsw i8 %A, %B
%I = mul nsw i8 %L, %C

%L = -4
%I = -128

%R = mul nsw i8 %B, %C
%I = mul nsw i8 %A, %R

%R = poison
%I = poison
Is This Valid?

\[ \%A = -1, \%B = 4, \%C = 32 \]

\[ \%L = \text{mul nsw i8} \%A, \%B \]
\[ \%I = \text{mul nsw i8} \%L, \%C \]

\[ \%R = \text{mul i8} \%B, \%C \]
\[ \%I = \text{mul nsw i8} \%A, \%R \]

\[ \%L = -4 \]
\[ \%I = -128 \]

\[ \%R = -128 \]
\[ \%I = \text{poison} \]
Is This Valid?

%A = -1, %B = 4, %C = 32

%L = mul nsw i8 %A, %B
%I = mul nsw i8 %L, %C

%R = mul i8 %B, %C
%I = mul i8 %A, %R

%L = -4
%I = -128

%R = -128
%I = -128
...but flags are also essential
Flags Aid Optimization

%C = mul i8 %A, %B
%R = sdiv i8 %C, %B

R = (A×B)÷B
Is R = A?
Flags Aid Optimization

%A = 100, %B = 100

%C = mul i8 %A, %B
%R = sdiv i8 %C, %B

%C = 10000
%R = 100
Flags Aid Optimization

%A = 100, %B = 100

%C = mul i8 %A, %B
%R = sdiv i8 %C, %B

Too big for i8!
Flags Aid Optimization

%C = mul i8 %A, %B
%R = sdiv i8 %C, %B

We could do this if we knew that A×B fits in 8 bits
Flags Aid Optimization

More context for why flags are helpful and where they are generated (C)

%C = mul nsw i8 %A, %B
%R = sdiv i8 %C, %B

We could do this if we knew that A×B fits in 8 bits
…which is just what NSW/NUW are for
Outline

• Motivation
• Introducing Alive
• Language Overview
• Automated Verification
• Code Generation
• Conclusion
Alive: Fast, Safe, Easy

- Write optimizations in LLVM-like DSL
- Automatically verify correctness
- Automatically generate C++ code

Partial translation

Name: sdiv exact
%a = sdiv exact %x, %y
%r = mul %a, %y
=>
%r = %x

Name: sdiv inexact
%a = sdiv %x, %y
%r = mul %a, %y
=>
%b = srem %x, %y
%r = sub %x, %b

Partial translation
Prove Optimizations Correct

Alive handles all the tricky corners of LLVM’s IR
Automatic Code Generation

Alive writes your InstCombine code for you
Why Alive?

- Use of formal methods is not new
  - CompCert—Formally verified C compiler
  - Vellvm—Formal semantics for LLVM in Coq
- Nuno Lopes described verifying InstCombine with SMT solvers last year
Lightweight Formal Methods

• Automation
  • Use SMT to avoid manual proofs
  • Use Alive to avoid writing SMT queries
• High-level specification language
Alive Language

Pre: C2 % (1<<C1) == 0
%s = shl nsw %A, C1
%r = sdiv %s, C2
=>
%r = sdiv %A, C2/(1<<C1)

• 1<<C1 divides C2
• rs = (A<<C1)÷C2
• rt = A÷(C2÷(1<<C1))
Pre: \( C2 \% (1 \ll C1) == 0 \)
\( \%s = \text{shl nsw } \%A, C1 \)
\( \%r = \text{sdiv } \%s, C2 \)

\( \Rightarrow \)
\( \%r = \text{sdiv } \%A, C2/(1 \ll C1) \)
Alive Language

Pre: \( C2 \mod (1 \ll C1) = 0 \)
\( \%s = \text{shl\ nsw\ } \%A,\ C1 \)
\( \%r = \text{sdiv}\ \%s,\ C2 \)
\( \implies \%r = \text{sdiv}\ \%A,\ C2/(1\ll C1) \)

Constants

- Represent arbitrary immediate values
- Expressions permitted in target and precondition
Predicates and Functions

- Predicates can be used in precondition
  - May invoke heuristics in LLVM, e.g., WillNotOverflowSignedAdd

- Functions extend constant language
  - Most apply only to constant values, e.g., umax
  - width(%x) returns the bit width of any value
Predicates and Functions

Pre: $C < 0 \land \text{isPowerOf2}(\text{abs}(C))$

$\%0p0 = \text{add} \ %Y, \ C1$

$\%r = \text{mul} \ %0p0, \ C$

$\Rightarrow$

$\%\text{sub} = \text{sub} \ -C1, \ %Y$

$\%r = \text{mul} \ %\text{sub}, \ \text{abs}(C)$
Checking Correctness

- SMT (“Satisfiability Modulo Theories”)
  - Generalizes SAT solving
  - Additional theories for integers, bit vectors, etc.
- Undecidable in general
- Efficient in practice
  - Z3, Boolector, CVC4, etc.
Type Checking

- Translate type constraints to SMT
  - Binary operation arguments and answer have same type
  - Trunc result has fewer bits than argument, etc.
- Find and test all solutions to constraints
Checking Correctness

- Need to show that target *refines* source
- Target’s behavior undefined only when source’s is
- Target returns poison only when source does
- For all other inputs, target and source yield same result
Checking Correctness

- SMT finds satisfying instances
- Phrase queries as negations:
  - “Find an input where the source is defined but the target is undefined”
- Z3 either finds a counterexample, or shows that none exists
Checking Correctness

- Translation uses Z3’s theory of bitvectors
  - Sized, 2s-complement arithmetic
- undef uses theory of quantifiers
  - Optimization must hold for all possible values
Pre: C2 % (1<<C1) == 0
%s = shl nsw %A, C1
%r = sdiv %s, C2
=>
%r = sdiv %A, C2/(1<<C1)
(declare-fun C1 () (_ BitVec 4))
(declare-fun C2 () (_ BitVec 4))
(declare-fun %A () (_ BitVec 4))
(assert
  (and
    (and (distinct C2 (_ bv0 4)) true)
    (or
      (and (distinct (bvshl %A C1) (_ bv8 4)) true)
      (and (distinct C2 (_ bv15 4)) true)
    )
  )
(bvult C1 (_ bv4 4))
(= (bvashr (bvshl %A C1) C1) %A)
(= (bvsrem C2 (bvshl (_ bv1 4) C1)) (_ bv0 4))
(assert
  (distinct
    (bvsdiv (bvshl %A C1) C2)
    (bvsdiv %A (bvsdiv C2 (bvshl (_ bv1 4) C1)))
  )
  true
)
)
(check-sat)
(declare-fun C1 () (_ BitVec 4))
(declare-fun C2 () (_ BitVec 4))
(declare-fun %A () (_ BitVec 4))
(assert
  (and
    (and (distinct C2 (_ bv0 4)) true)
    (or
      (and (distinct (bvshl %A C1) (_ bv8 4)) true)
      (and (distinct C2 (_ bv15 4)) true)
    )
    (bvult C1 (_ bv4 4))
    (= (bvsrem C2 (bvshl (_ bv1 4) C1)) (_ bv0 4))
    (and
      (distinct
        (bvsdiv (bvshl %A C1) C2)
        (bvsdiv %A (bvsdiv C2 (bvshl (_ bv1 4) C1)))
      )
      true
    )
  )
)(check-sat)
Is This Valid?

Pre: C2 % (1 << C1) == 0
%s = shl nsw %A, C1
%r = sdiv %s, C2
=>
%r = sdiv %A, C2/(1 << C1)
Is This Valid?

Pre: \(C2 \% (1 \ll C1) == 0\)
\(\%s = \text{shl nsw} \%A, C1\)
\(\%r = \text{sdiv} \%s, C2\)

=>
\(\%r = \text{sdiv} \%A, C2/(1 \ll C1)\)

ERROR: Mismatch in values of i4 \%r

Example:
\(\%A\) i4 = 0xF (15, -1)
\(C1\) i4 = 0x3 (3)
\(C2\) i4 = 0x8 (8, -8)
\(\%s\) i4 = 0x8 (8, -8)

Source value: 0x1 (1)
Target value: 0xF (15, -1)

Alive finds a counterexample
Is This Valid?

Pre: C2 \%(1\ll C1) == 0
%s = shl nsw %A, C1
%r = sdiv %s, C2
=>
%r = sdiv %A, C2/(1\ll C1)

ERROR: Mismatch in values of i4 \%r

Example:
%A i4 = 0xF (15, -1)
C1 i4 = 0x3 (3)
C2 i4 = 0x8 (8, -8)
%s i4 = 0x8 (8, -8)
Source value: 0x1 (1)
Target value: 0xF (15, -1)

Alive finds a counterexample

%r = (%A\ll C1) / C2  %r = %A / (C2/(1\ll C1))
Is This Valid?

Pre: C2 \% (1<<C1) == 0
\%s = shl nsw \%A, C1
\%r = sdiv \%s, C2
=>
\%r = sdiv \%A, C2/(1<<C1)

ERROR: Mismatch in values of i4 \%r

Example:
\%A i4 = 0xF (15, -1)
C1 i4 = 0x3 (3)
C2 i4 = 0x8 (8, -8)
\%s i4 = 0x8 (8, -8)
Source value: 0x1 (1)
Target value: 0xF (15, -1)

Alive finds a counterexample

\%r = (-1<< 3) / -8 \%r = -1 / (-8/(1<< 3))
Is This Valid?

Pre: C2 % (1<<C1) == 0
%s = shl nsw %A, C1
%r = sdiv %s, C2
=>
%r = sdiv %A, C2/(1<<C1)

ERROR: Mismatch in values of i4 %r

Example:
%A i4 = 0xF (15, -1)
C1 i4 = 0x3 (3)
C2 i4 = 0x8 (8, -8)
%s i4 = 0x8 (8, -8)
Source value: 0x1 (1)
Target value: 0xF (15, -1)

Alive finds a counterexample

%r = -8 / -8
%r = -1 / (-8/-8)
Is This Valid?

Pre: $C_2 \% (1 \ll C_1) == 0$
$s = \text{shl nsw} \ A, C_1$
$r = s\text{div} \ s, C_2$
$=>$
$r = s\text{div} \ A, C_2/(1 \ll C_1)$

ERROR: Mismatch in values of i4 %r

Example:
%A i4 = 0xF (15, -1)
C1 i4 = 0x3 (3)
C2 i4 = 0x8 (8, -8)
%s i4 = 0x8 (8, -8)
Source value: 0x1 (1)
Target value: 0xF (15, -1)

Alive finds a counterexample

$r = \begin{cases} 1 & \text{if } r = -1 / 1 \end{cases}$
Is This Valid?

Pre: C2 % (1<<C1) == 0
%s = shl nsw %A, C1
%r = sdiv %s, C2
=>
%r = sdiv %A, C2/(1<<C1)

ERROR: Mismatch in values of i4 %r

Example:
%A i4 = 0xF (15, -1)
C1 i4 = 0x3 (3)
C2 i4 = 0x8 (8, -8)
%s i4 = 0x8 (8, -8)
Source value: 0x1 (1)
Target value: 0xF (15, -1)

Alive finds a counterexample

%r = (-1<< 3) / -8
%r = -1 / (-8/(1<< 3))
Is This Valid?

Pre: $C_2 \% (1 \ll C_1) = 0$
$s = \text{shl} \ nsw \ %A, \ C_1$
$r = \text{sdiv} \ s, \ C_2$

$\Rightarrow$
$r = \text{sdiv} \ %A, \ C_2/(1\ll C_1)$

ERROR: Mismatch in values of i4 $r$

Example:
$%A \ i4 = 0xF \ (15, \ -1)$
$C_1 \ i4 = 0x3 \ (3)$
$C_2 \ i4 = 0x8 \ (8, \ -8)$
$s \ i4 = 0x8 \ (8, \ -8)$

Source value: 0x1 (1)
Target value: 0xF (15, -1)

Alive finds a counterexample

$r = (\ -1\ll 3 \ / \ -8 \ \ r = -1 \ / \ (-8/(1\ll 3)))$
Is This Valid?

Pre: C2 % (1<<C1) == 0
%s = shl nsw %A, C1
%r = sdiv %s, C2
=>
%r = sdiv %A, C2/(1<<C1)

ERROR: Mismatch in values of i4 %r

Example:
%A i4 = 0xF (15, -1)
C1 i4 = 0x3 (3)
C2 i4 = 0x8 (8, -8)
%s i4 = 0x8 (8, -8)
Source value: 0x1 (1)
Target value: 0xF (15, -1)

Alive finds a counterexample

1<<C1 wraps when C1 = width(C1)−1
This Is Valid

Pre: \( C2 \% (1<<C1) == 0 \) \&\& \( C1 != \text{width}(C1) - 1 \)
%\( s \) = shl nsw %A, C1
%\( r \) = sdiv %s, C2
  =>
%\( r \) = sdiv %A, C2/(1<<C1)

Possibly too conservative?
This Is Also Valid

Pre: $C_2 \% (1<<C_1) == 0$
%s = shl nsw %A, C1
%r = sdiv %s, C2
=>
%r = sdiv %A, C2>>C1

ashr never wraps sign
This Is Also Valid

Pre: $C_2 \% (1<<C_1) == 0$
%s = shl nsw %A, C1
%r = sdiv %s, C2
=>
%r = sdiv %A, C2>>C1

$C_2/(1<<C_1)$
$C_2>>C1$
This Is Also Valid

Pre: C2 % (1<<C1) == 0
%s = shl nsw %A, C1
%r = sdiv %s, C2
=>
%r = sdiv %A, C2>>C1

-8/(1<< 3)
-8>>3
This Is Also Valid

Pre: C2 % (1<<C1) == 0
%s = shl nsw %A, C1
%r = sdiv %s, C2
=>
%r = sdiv %A, C2>>C1

-8/-8
-1
Alive Is Already Useful

- We have uncovered several incorrect optimizations
  - PR20186, PR21243, PR21244, PR21245, PR21255, PR21256, PR21274
- Alive can check whether proposed fixes are correct
- Alive has helped improve several patches

Goal is to make sure new transformations are correct. Emphasize value for developers
Code Generation

• Translating Alive to LLVM Source is mechanical
  • …so let’s let our machines do it

• Avoid mistranslations of Alive to C++

• Alive is an order of magnitude smaller than InstCombine
Pre: C1 & C2 == 0 && MaskedValueIsZero(%V2, ~C1)
%A  = or %B, %V2
%op0 = and %A, C1
%op1 = and %B, C2
%r  = or %op0, %op1
    =>
%A  = or %B, %V2
%r  = and %A, (C1 | C2)
Value *op0, *op1, *B, *A, *V2;
ConstantInt *C1, *C2;

if (match(I, m_Or(m_Value(op0), m_Value(op1)))
    && match(op1, m_And(m_Value(B), m_ConstantInt(C2)))
    && match(op0, m_And(m_Value(A), m_ConstantInt(C1)))
    && match(A, m_Or(m_Specific(B), m_Value(V2)))
    && (C1->getValue() & C2->getValue()) == 0
    && MaskedValueIsZero(V2, ~C1->getValue())
{
    Value *r =
        BinaryOperator::CreateAnd(A, ConstantExpr::getOr(C1, C2), "", I);
    I->replaceAllUsesWith(r);
    return true;
}
Mostly Straightforward

- Instructions in source use LLVM’s pattern matching
- Instructions in target become constructors
- Constants become APInt or ConstantInt as needed
- Precondition and constant expressions map recursively to LLVM functions
- There are a few tricky bits…
Target creates constraints

- Target is well-typed only if $\text{type}(\%a) = \text{type}(\%b)$
- Source is always well-typed
- Alive introduces an additional check

\[
\begin{align*}
\%ax &= \text{zext } \%a \\
\%bx &= \text{zext } \%b \\
\%r &= \text{and } \%ax, \%bx \\
&\quad \Rightarrow \\
\%c &= \text{and } \%a, \%b \\
\%r &= \text{zext } \%c
\end{align*}
\]
Explicit Types in Target

- Literals and conversions require explicit types
- Alive matches target types to source values
Repeated Source Values

- Variables can occur multiple times in source
- Alive uses \texttt{m\_Specific} when possible
- Otherwise, introduces a dummy variable and an equality constraint

\begin{verbatim}
%r = add %a, %a
=>
%r = shl %a, 1
\end{verbatim}
Future Work

• Fill in missing parts
  • More types (e.g., floating point)
  • More predicates
• Parameterize on instructions
• Deduce flag placement
Let Alive Work for You

- Express optimizations in high-level DSL
- Automatically detect errors
- Use generated code in LLVM
- Open source (https://github.com/nunoplopes/alive)