Iago: AI Driven Superoptimization for LLVM

Iago 

Manasij Mukherjee
NVIDIA

John Regehr
University of Utah

Talk Organization

Motivation — Why solve this problem?

Design — What worked and what doesn't?

Results — How well does it work?

What is Superoptimization?

Can a piece of code be improved?

Specifically, by **searching** for a refinement.

$$\neg A \wedge \neg B \Rightarrow$$

```
define i32 @foo(i32 %x) {
entry:
  %a = and i32 %x, 1431655765
  %b = lshr i32 %x, 1
  %c = and i32 %b, 1431655765
  %d = add i32 %a, %c
  %e = and i32 %d, 858993459
  %f = lshr i32 %d, 2
  %g = and i32 %f, 858993459
  %h = add i32 %e, %g
  %i = and i32 %h, 252645135
  %j = lshr i32 %h, 4
  %k = and i32 %j, 252645135
  %l = add i32 %i, %k
  %m = and i32 %l, 16711935
  %n = lshr i32 %l, 8
  %o = and i32 %n, 16711935
  %p = add i32 %m, %o
  %q = and i32 %p, 65535
  %r = lshr i32 %p, 16
  %s = and i32 %r, 65535
  %t = add i32 %q, %s
  ret i32 %t
}
```

Searching for a refinement

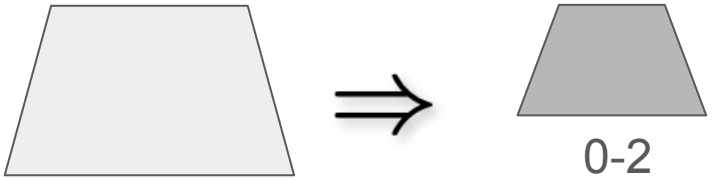
Enumerate a bunch of candidates

Attempt to verify each

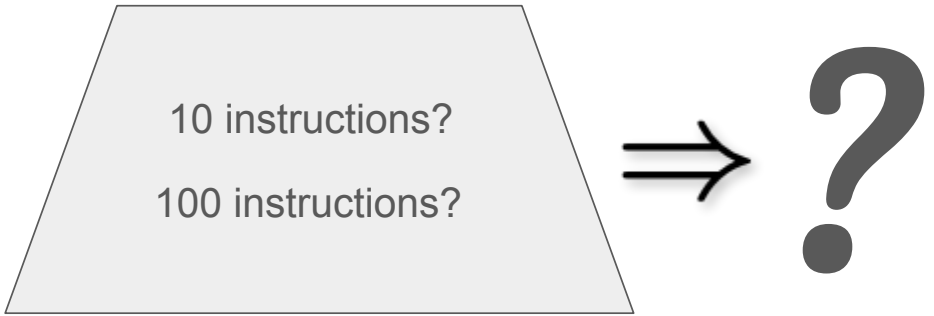
Superoptimizing sqlite3 (53.94k lines of code) takes ~24 hours with 128 cores with ***Souper***

What's on the table?

What we usually put into InstCombine today:

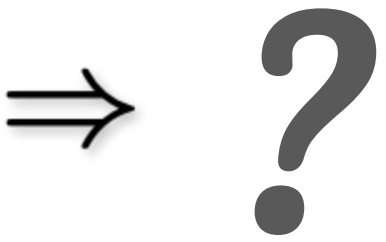


One off pattern in your important code that is not generally useful



```
define i32 @foo(i32 %x) {
entry:
  %a = and i32 %x, 1431655765
  %b = lshr i32 %x, 1
  %c = and i32 %b, 1431655765
  %d = add i32 %a, %c
  %e = and i32 %d, 858993459
  %f = lshr i32 %d, 2
  %g = and i32 %f, 858993459
  %h = add i32 %e, %g
  %i = and i32 %h, 252645135
  %j = lshr i32 %h, 4
  %k = and i32 %j, 252645135
  %l = add i32 %i, %k
  %m = and i32 %l, 16711935
  %n = lshr i32 %l, 8
  %o = and i32 %n, 16711935
  %p = add i32 %m, %o
  %q = and i32 %p, 65535
  %r = lshr i32 %p, 16
  %s = and i32 %r, 65535
  %t = add i32 %q, %s
  ret i32 %t
}
```

Do we really need to optimize long sequences?



Superoptimization examples

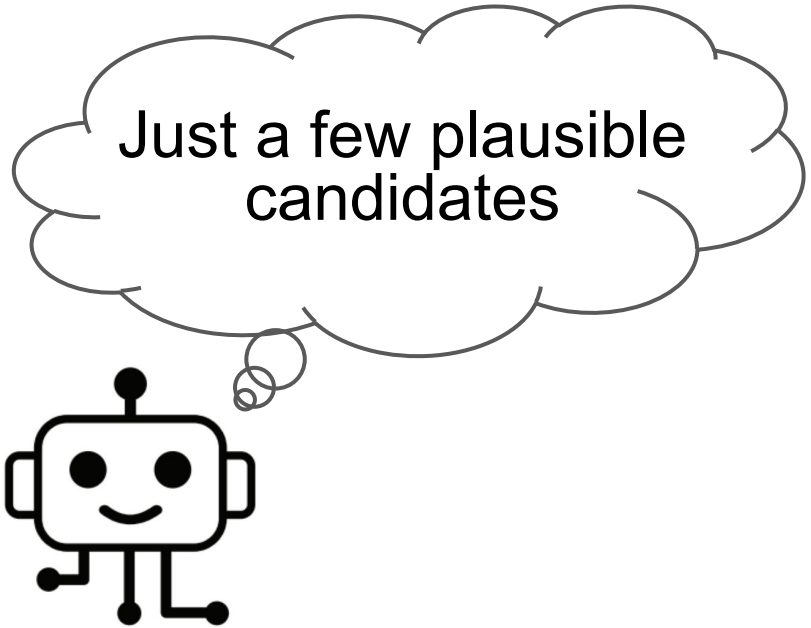
```
define i32 @foo(i32 %x) {
entry:
  %a = and i32 %x, 1431655765
  %b = lshr i32 %x, 1
  %c = and i32 %b, 1431655765
  %d = add i32 %a, %c
  %e = and i32 %d, 858993459
  %f = lshr i32 %d, 2
  %g = and i32 %f, 858993459
  %h = add i32 %e, %g
  %i = and i32 %h, 252645135
  %j = lshr i32 %h, 4
  %k = and i32 %j, 252645135
  %l = add i32 %i, %k
  %m = and i32 %l, 16711935
  %n = lshr i32 %l, 8
  %o = and i32 %n, 16711935
  %p = add i32 %m, %o
  %q = and i32 %p, 65535
  %r = lshr i32 %p, 16
  %s = and i32 %r, 65535
  %t = add i32 %q, %s
  ret i32 %t
}
```

```
ctpop %x
```

```
define i32 @src(i1 %c) {
  %8 = zext i1 %c to i32
  %9 = select i1 %c, i32 2, i32 0
  %10 = or i32 %8, %9
  %11 = select i1 %c, i32 4, i32 0
  %12 = or i32 %10, %11
  %13 = select i1 %c, i32 8, i32 0
  %14 = or i32 %12, %13
  %15 = select i1 %c, i32 256, i32 0
  %16 = or i32 %14, %15
  %17 = select i1 %c, i32 512, i32 0
  %18 = or i32 %16, %17
  %19 = select i1 %c, i32 1024, i32 0
  %20 = or i32 %18, %19
  %21 = select i1 %c, i32 2048, i32 0
  %22 = or i32 %20, %21
  ret i32 %22
}
```

```
select i1 %c, i32 3855, i32 0
```

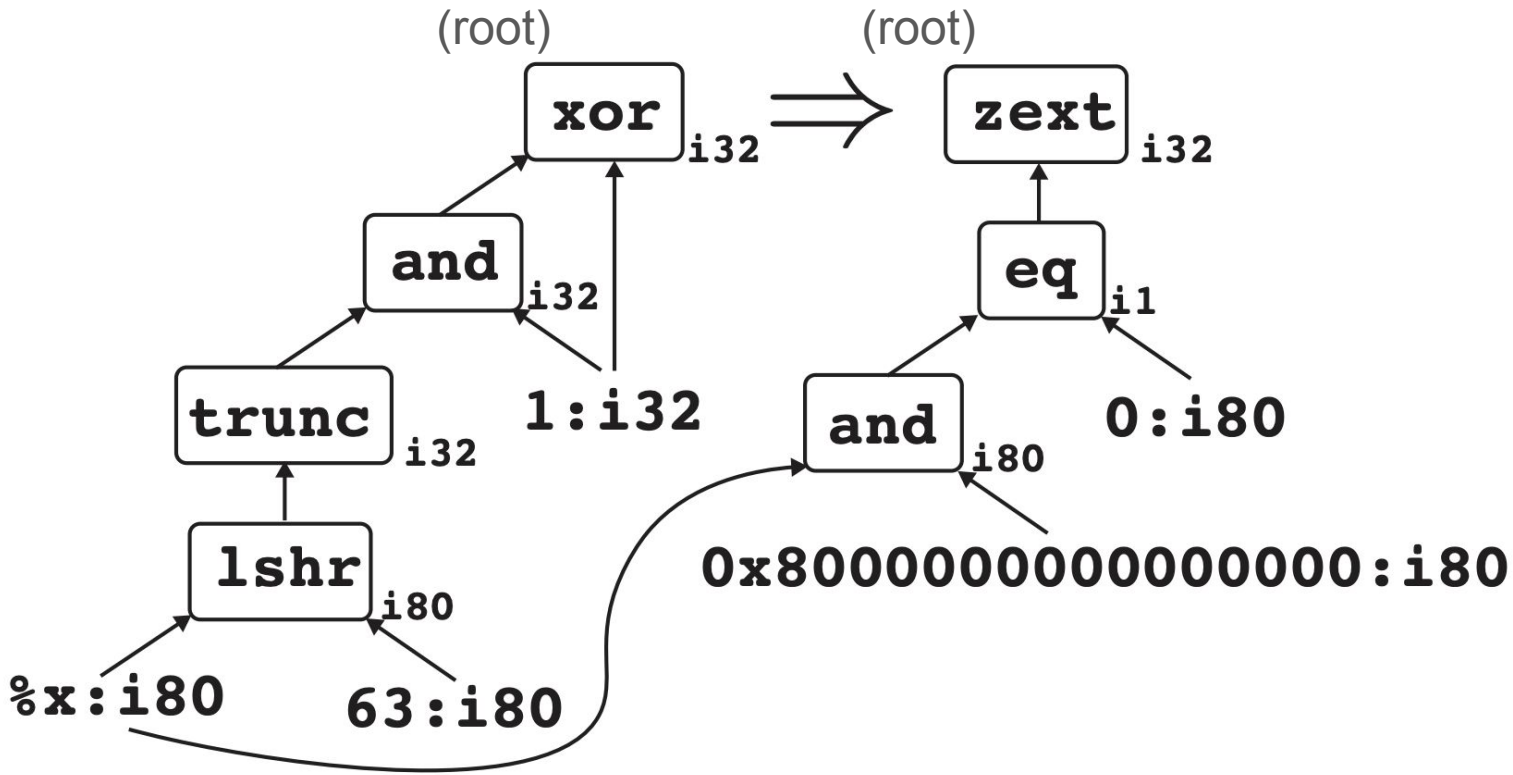
Why replace the enumeration with AI?



Reach far deeper into the search space

Attempt to verify each

Example - guess what this is!



Superoptimization → Compilers

Can a piece of code be improved?

Can we improve the compiler such that -

Hydra

it improves this piece of code?

... it improves other, similar, pieces of code?

... and does not regress anything else?

Talk Organization

Motivation — Why solve this problem?

Design — What worked and what doesn't?

Results — How well does it work?

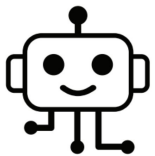
Baseline: just asking the LLM!

$$\neg A \wedge \neg B \Rightarrow ?$$

The expression you've given seems to be an incomplete logical expression. The "=>" symbol typically indicates implication, suggesting that ...

Two significant issues

$$\neg A \wedge \neg B \Rightarrow ?$$



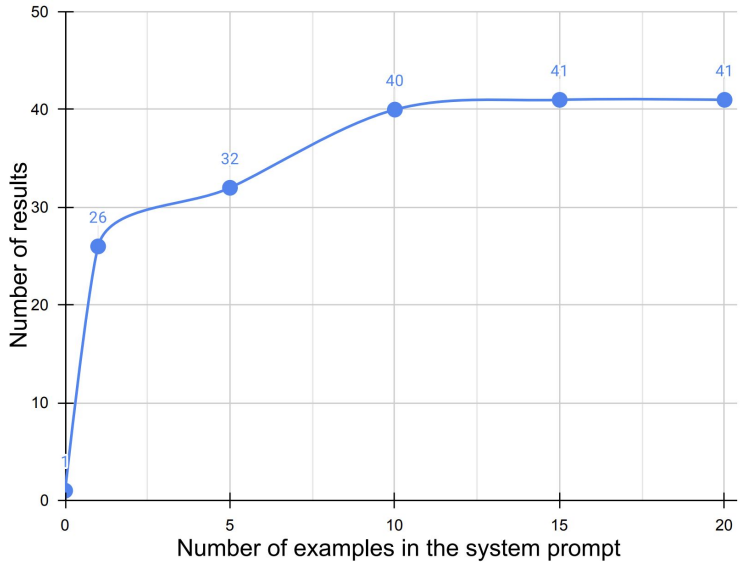
The expression you've given seems to be an incomplete logical expression. The "=>" symbol typically indicates implication, suggesting that ...

Wrong syntax!

Wrong semantics!

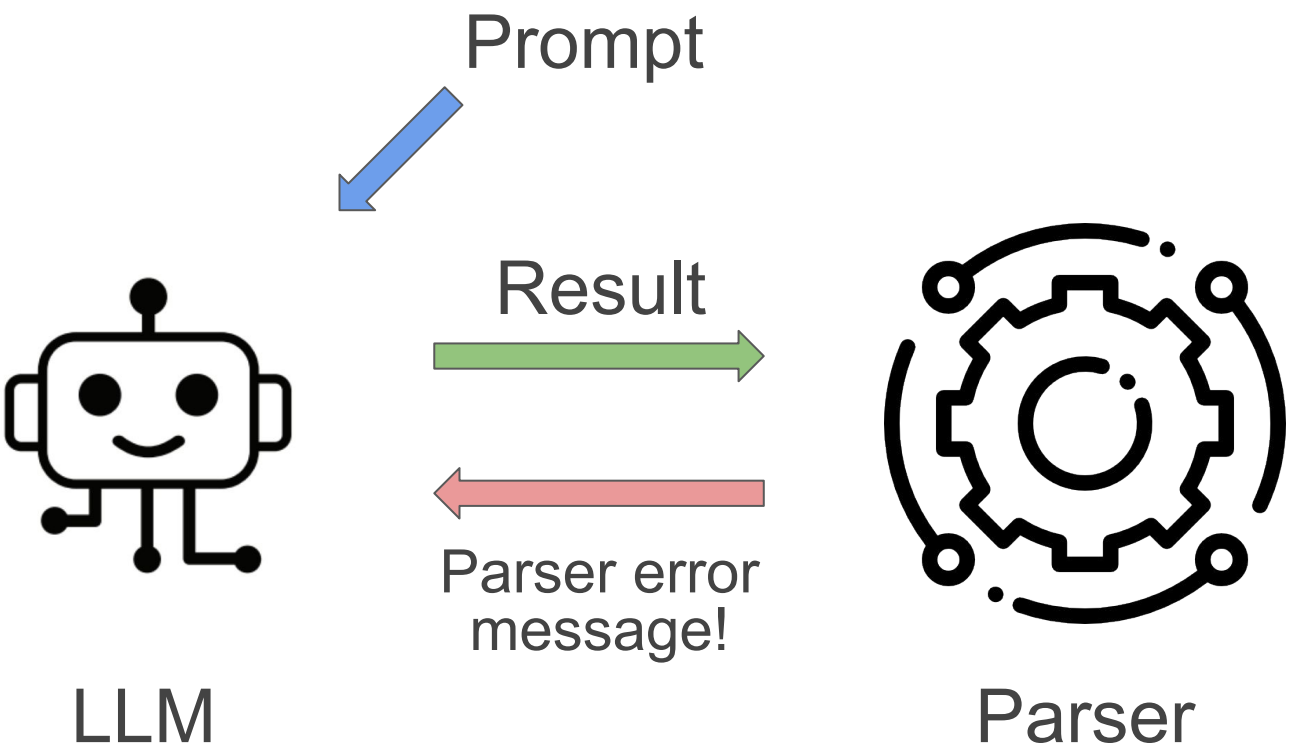
Iago's System Prompt

- Explain the idea of peephole optimizations
- Describe available operations
- Negative statements about what to avoid
- Examples



No correctness guarantee from an LLM

Getting the right syntax



Fixing seemingly consistent issues

LLM results kept redefining values

```
%x = add i32 %a, %b
%x = add i32 %x, 1
```

```
%x = add i32 %a, %b
%x' = add i32 %x, 1
```

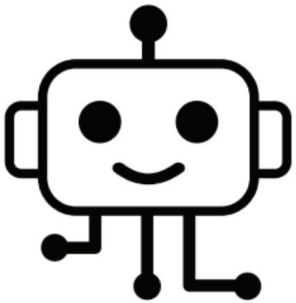
Wrong types

```
%x = or i32 %a, %b
%c = select i32 %x, ...
```

Insert appropriate icmp or trunc

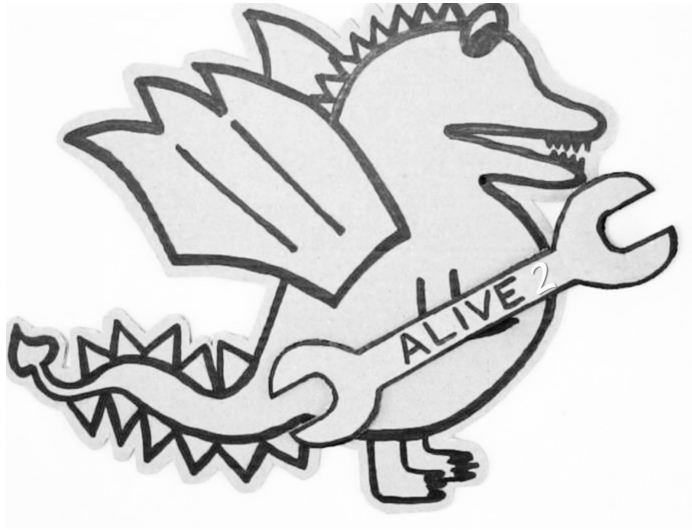
What about correctness?

Prompt



Result

Counter-example

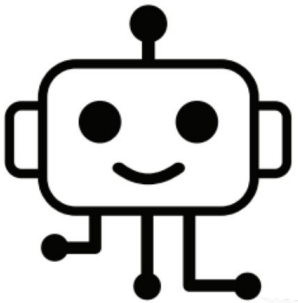


LLM

Alive2

Does not work, at all.

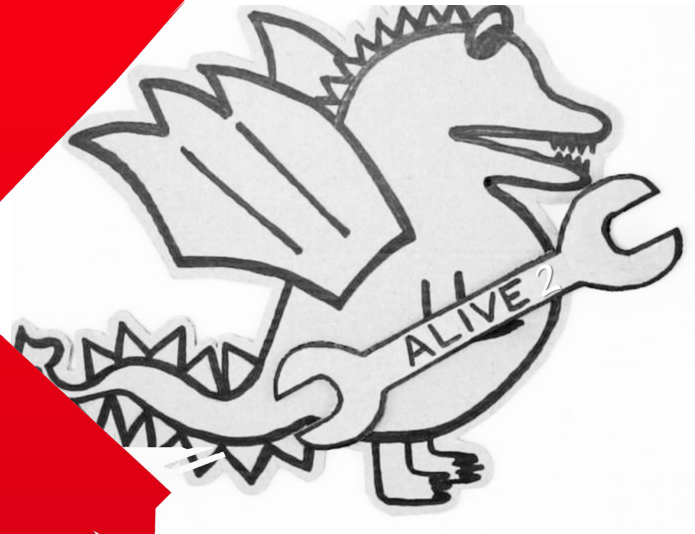
LLMs do not seem to understand counterexamples



LLM

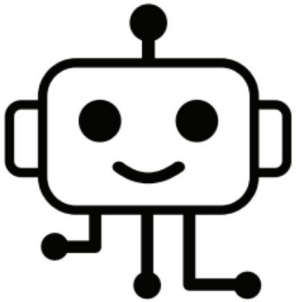
Prompt

-exam



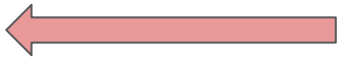
Alive2

Works, sort of...



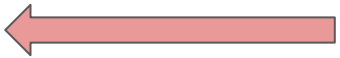
LLM

BLAH

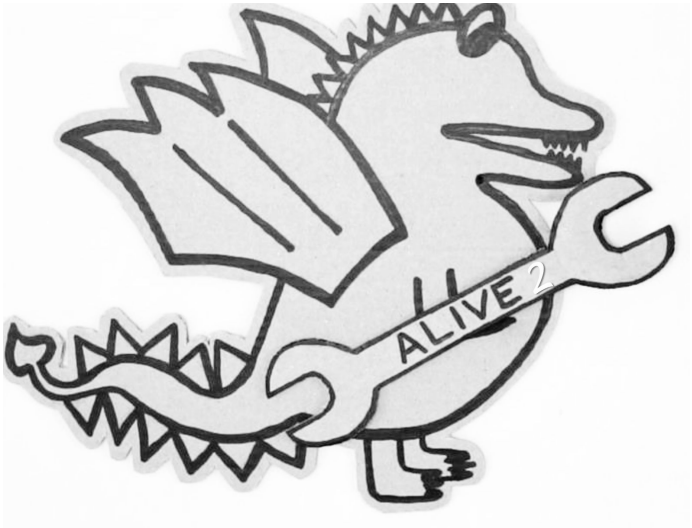


Just try again!

BLAAAAH



Just try again!



Alive2

Can we do better?

Formal Methods to the rescue again!

$$\neg A \wedge \neg B \Rightarrow ?$$

No correctness guarantee from an LLM

Verify Results

with

Alive2 + Z3

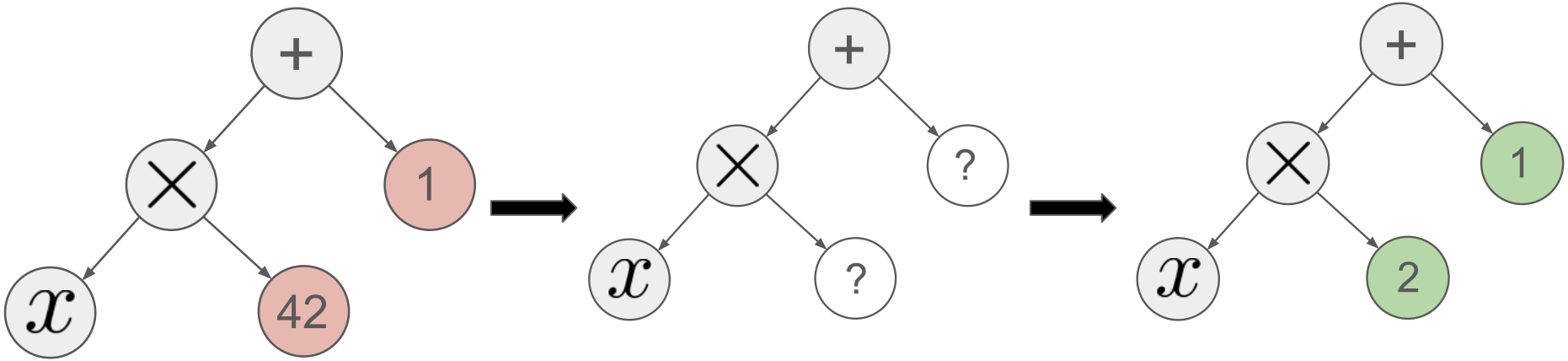
Fix Results

with

CEGIS

Fixing Incorrect Results

Extract a sketch



Fill in the blanks with an SMT solver

How do we fill in the blanks with an SMT solver?

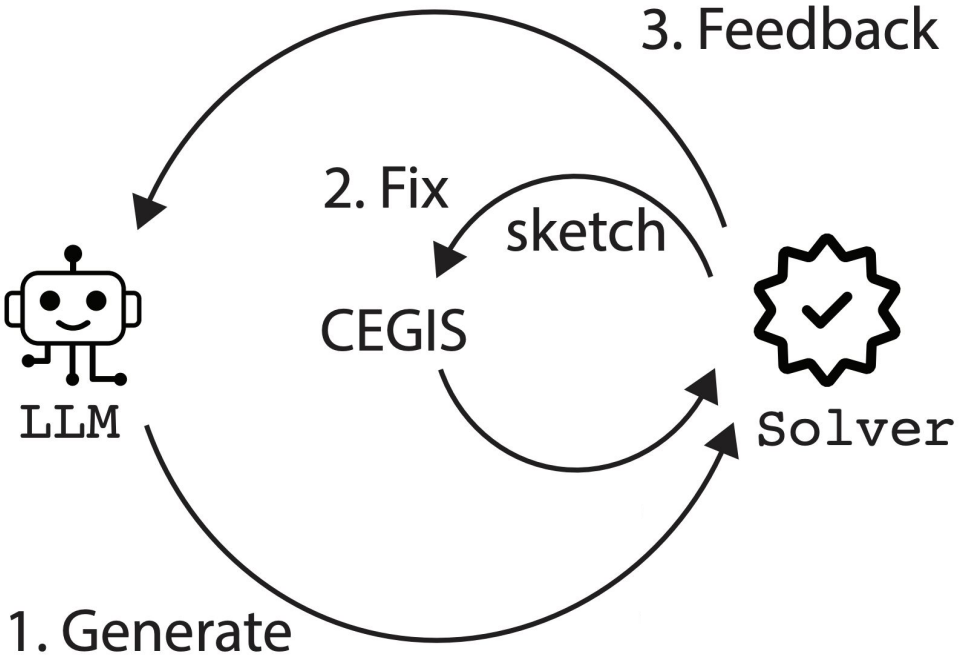
This is the logical formula in question

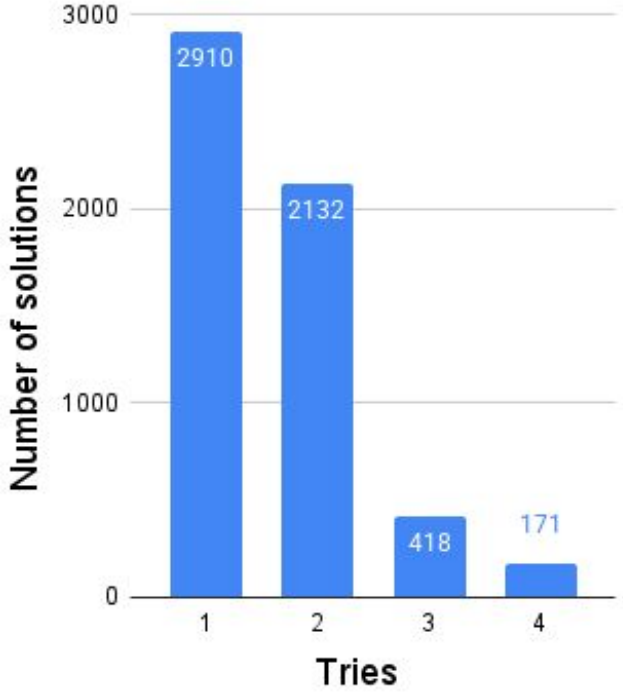
$$\exists C. \forall x. f(x, C) = g(x, C)$$

Counterexample Guided Inductive Synthesis

Solving Exists/Forall Problems With Yices [Dutertre]

Recap : Iago's Synthesis Loops



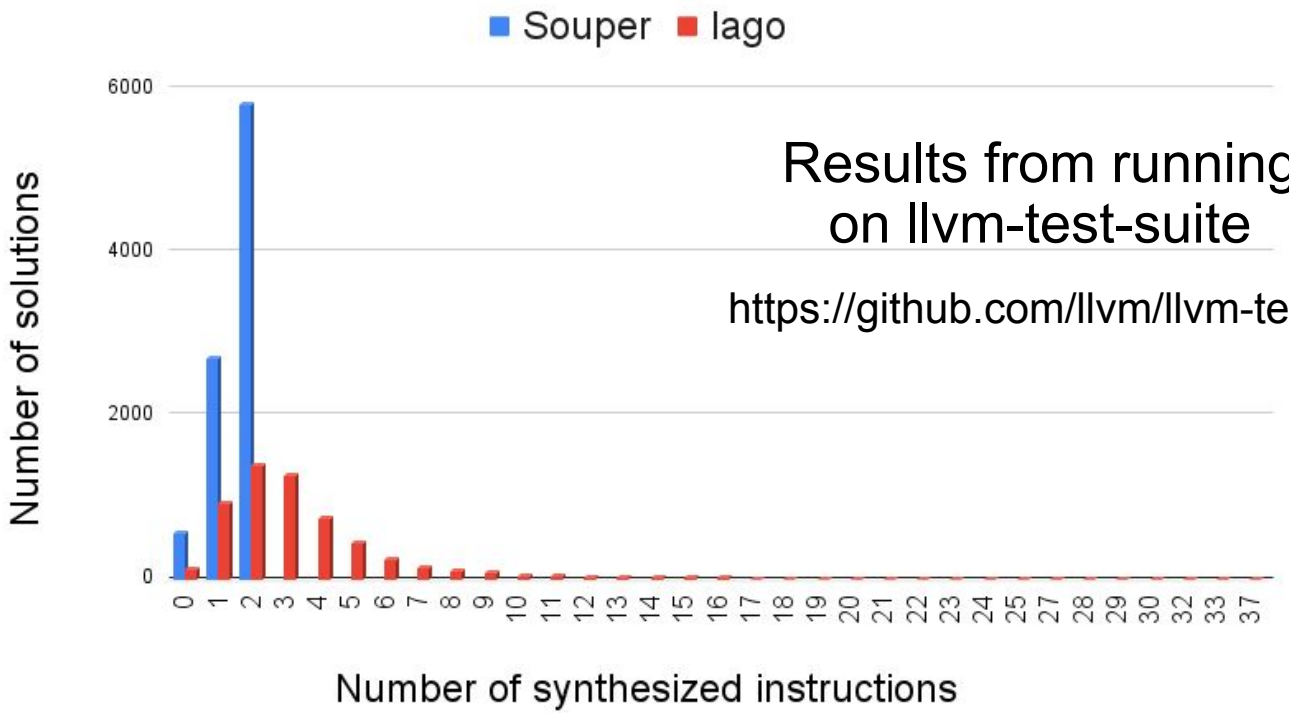


All results formally verified

What about completeness?

Can we guarantee finding a refinement if one exists ?

No.

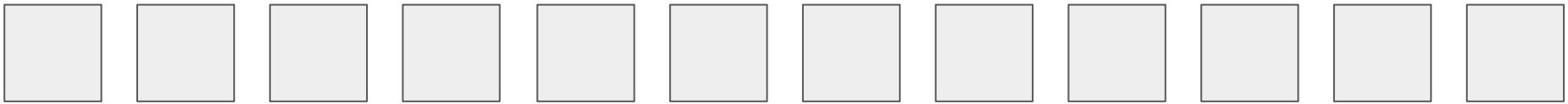


Results from running on llvm-test-suite

https://github.com/llvm/llvm-test-suite

Towards Completeness

First level of search tree, incomplete candidates.



Pruning, remove infeasible candidates

Fits in an LLM context window now!

Iago is sound but not complete.

Monkey's Paw

We have wished for :

Correct syntax

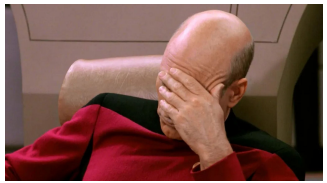
Must be a valid refinement

```
x == y ⊨

(x & ~y) == 0
```



```
x == y
```



```
%0:i64 = var
%1:i64 = var
%2:i64 = xor -1:i64, %1
%3:i64 = and %0, %2
%4:i1 = eq 0:i64, %3
infer %4
%5:i1 = eq %0, %1
pc %5 1:i1
result %5
```

Talk Organization

Motivation — Why solve this problem?

Design — What worked and what doesn't?

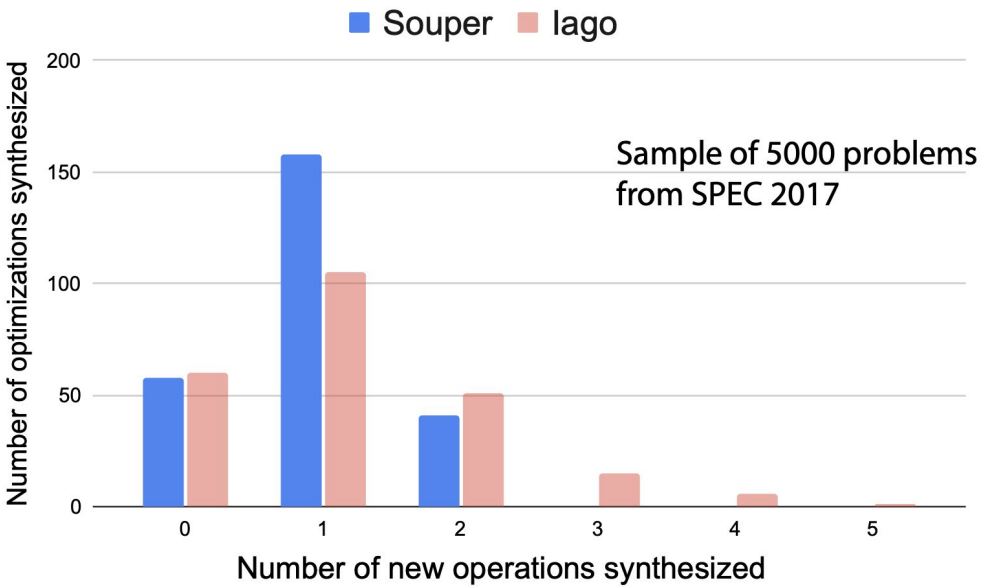
Results — How well does it work?

lago results summary

More complex optimizations, complements Souper.

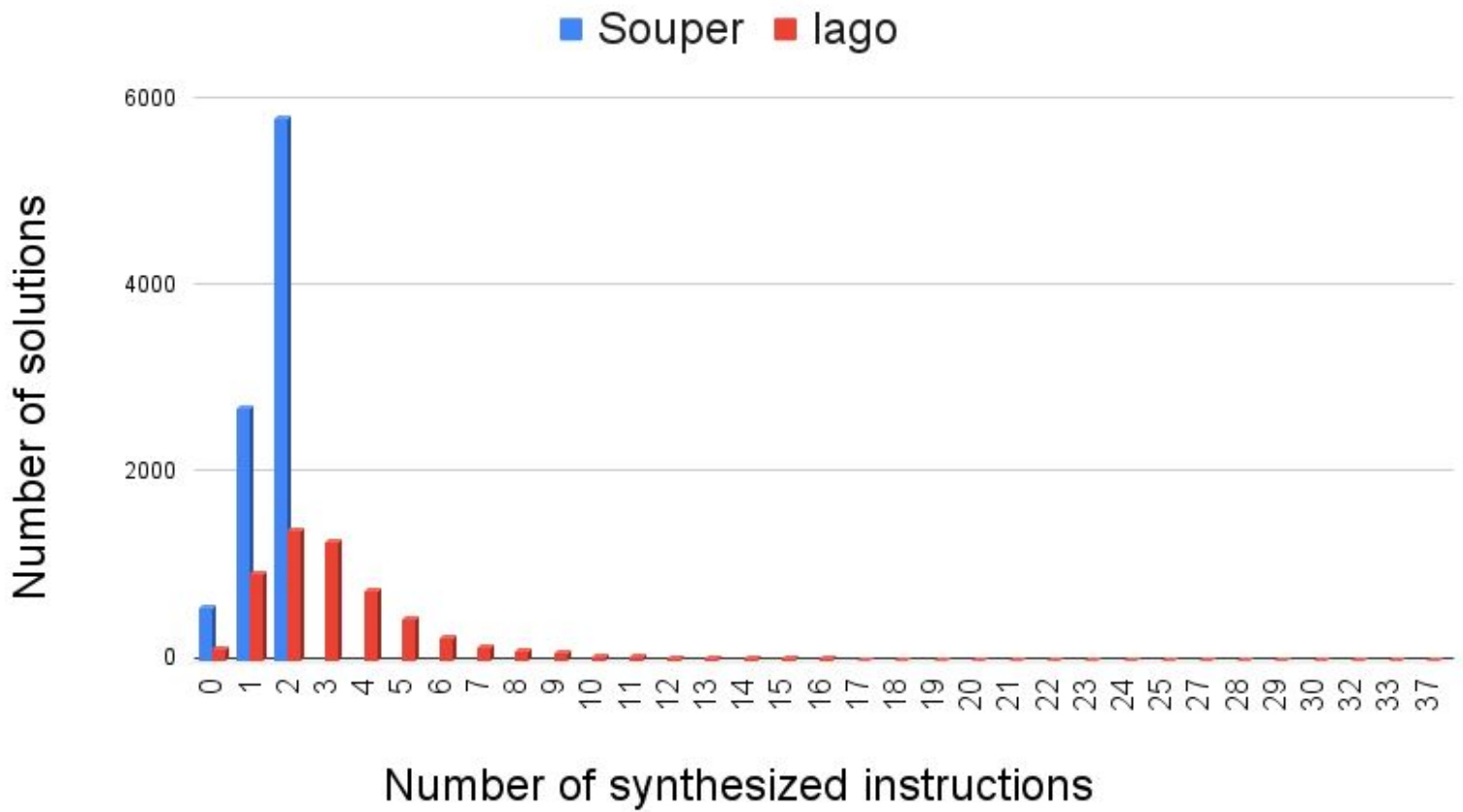
35.7% results are new!

7.4% fewer total results



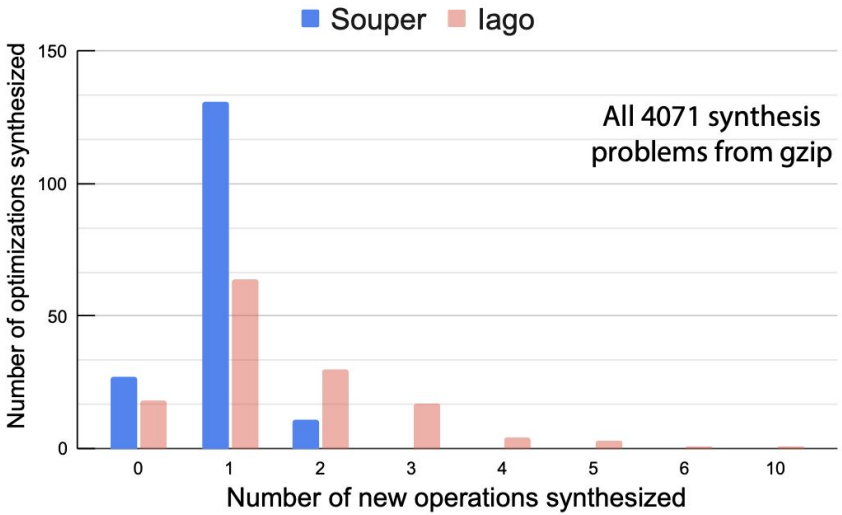
Results on llvm-test-suite

https://github.com/llvm/llvm-test-suite

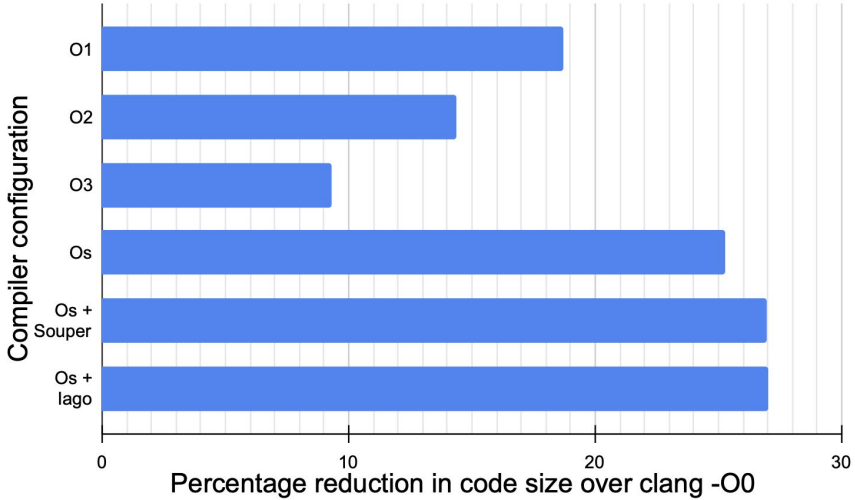


lago results summary

Finds optimizations in gzip a 30 year old codebase.



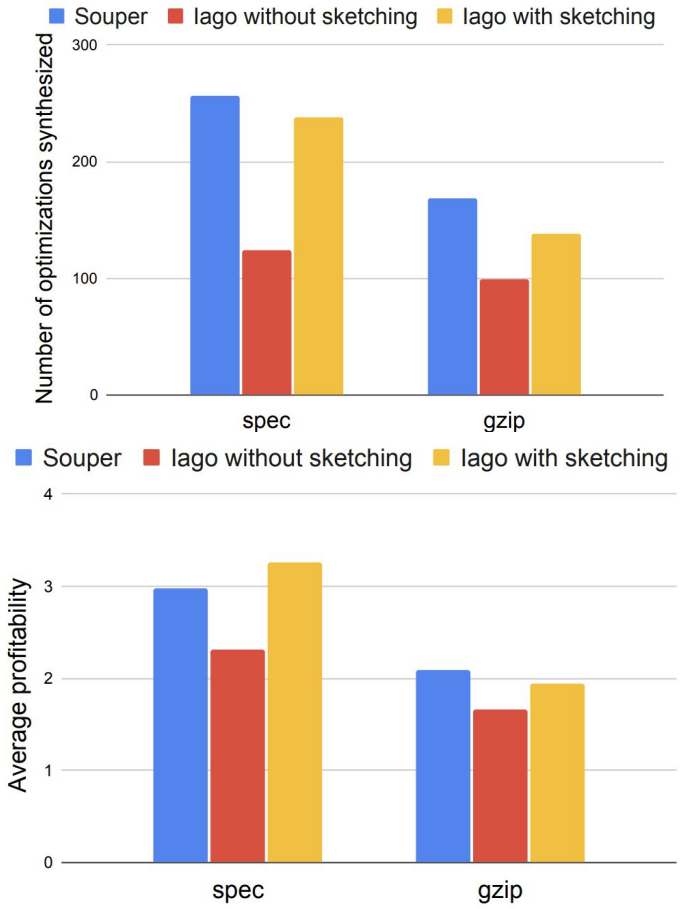
Matches Souper's code size reduction for gzip



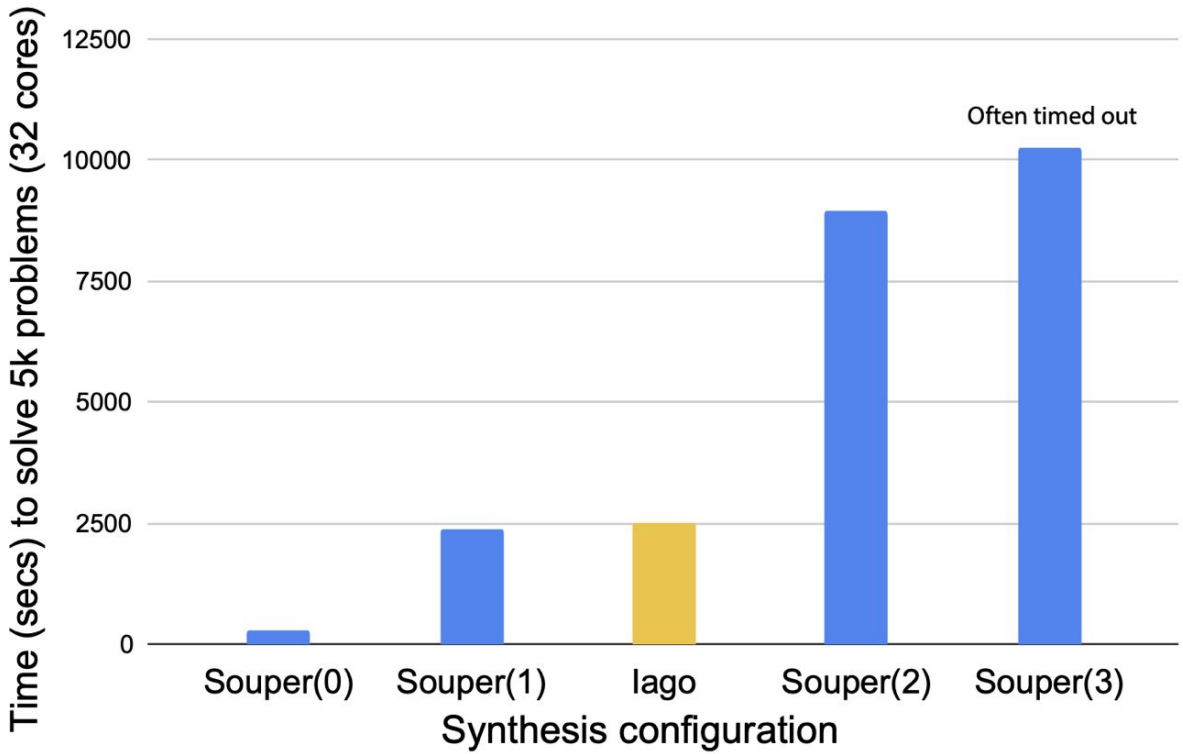
Does sketching help?

More results

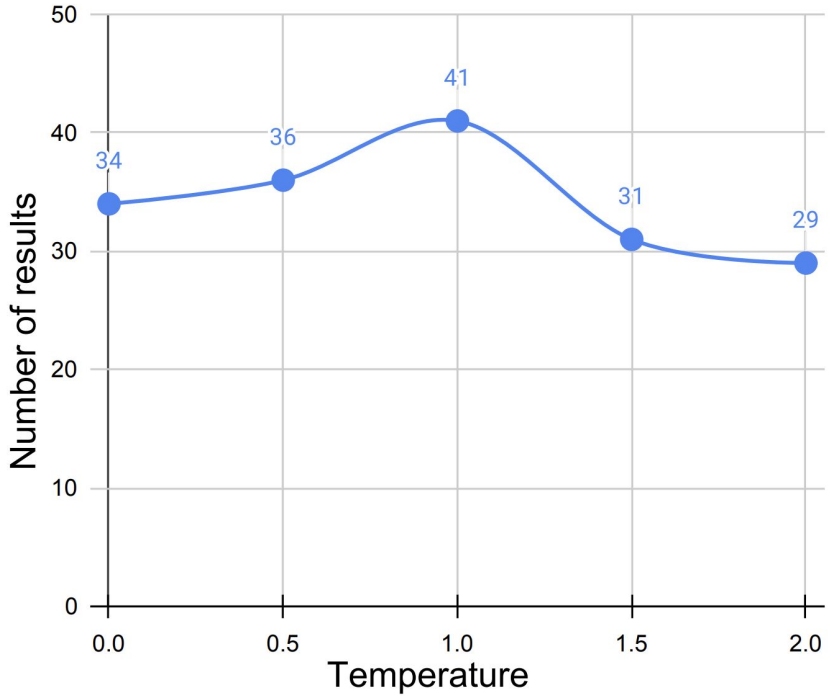
More profitable results



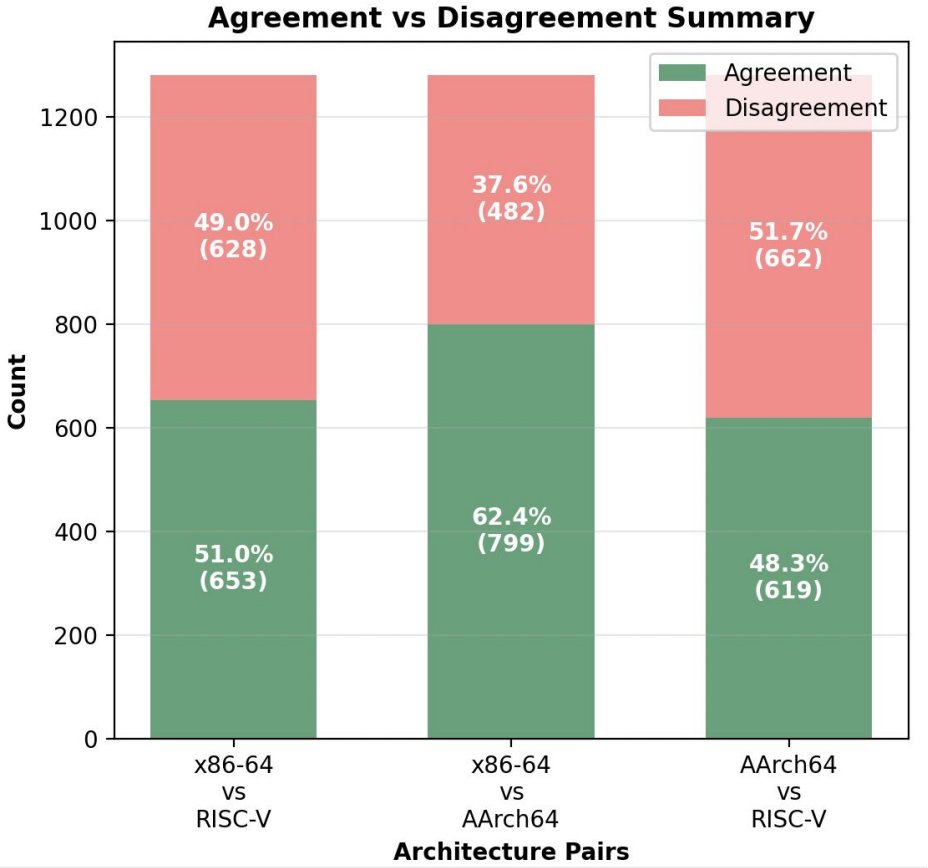
Is it faster than Souper?



GPT-4 Temperature Parameter



Do different backends agree on Iago's optimizations?




```
sub nuw i64 31, %x
```

→

```
xor i64 %x, 31
```

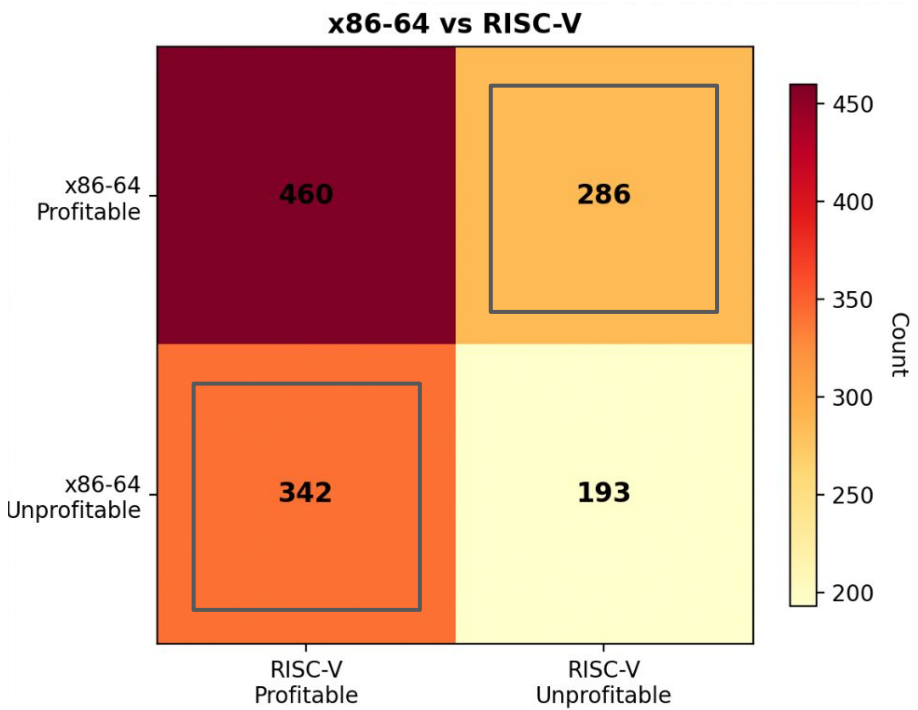
x86-64

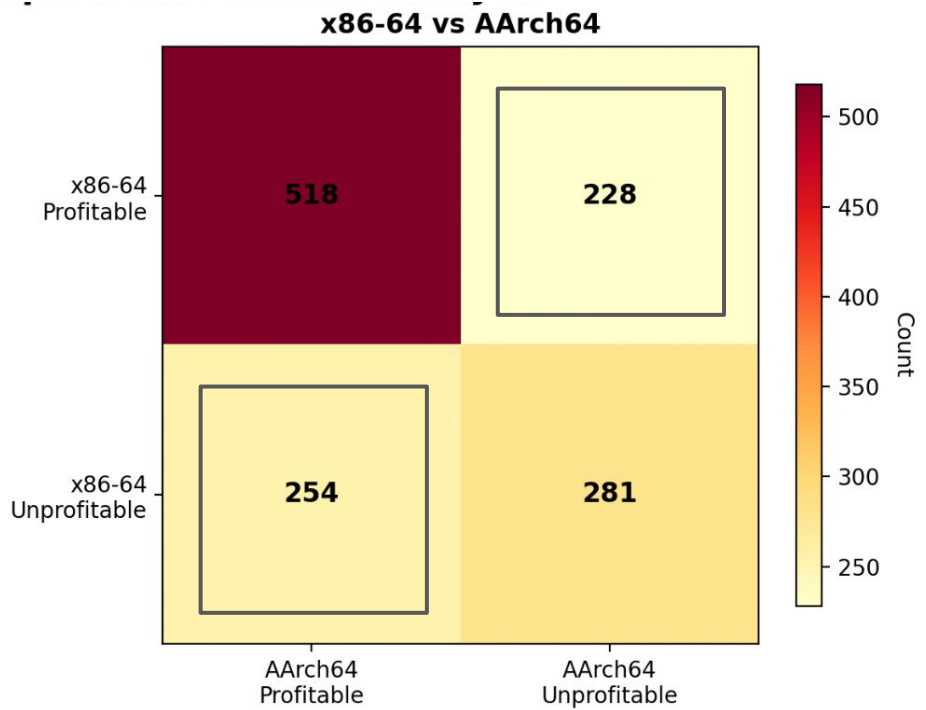
```
mov rax, rdi
xor rax, 31
```

riscv

```
xori a0, a0, 31
```

Breakdown - x86-64 vs RISC-V and x86-64 vs AArch64





Profitable in this context - produces fewer asm instructions

Takeaways implementing these optimizations

```
define i32 @src(i1 %c) {
  %8 = zext i1 %c to i32
  %9 = select i1 %c, i32 2, i32 0
  %10 = or i32 %8, %9
  %11 = select i1 %c, i32 4, i32 0
  %12 = or i32 %10, %11
  %13 = select i1 %c, i32 8, i32 0
  %14 = or i32 %12, %13
  %15 = select i1 %c, i32 256, i32 0
  %16 = or i32 %14, %15
  %17 = select i1 %c, i32 512, i32 0
  %18 = or i32 %16, %17
  %19 = select i1 %c, i32 1024, i32 0
  %20 = or i32 %18, %19
  %21 = select i1 %c, i32 2048, i32 0
  %22 = or i32 %20, %21
  ret i32 %22
}
```

Often multiple transforms in disguise

```
define i32 @tgt(i1 %c) #0 {
  %8 = select i1 %c, i32 3855, i32 0
  ret i32 %8
}
```

Takeaways

Takeaways implementing these optimizations

Conflicts with canonicalization

Termination issues!

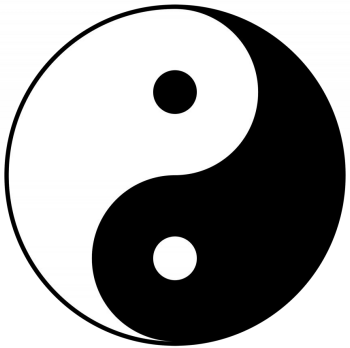
```
zext(xor 1:i1, b:i1)
  => 1 - zext(b)
```

Picking the right subset is tricky

Symbiosis between AI and Formal Methods

Great at generating text

Exhaustive synthesis



No correctness guarantees

Always correct

Some final - REDO

```
!(x0:i8 <u 9)
  |=
zext(((zext(x0) <<nw 3)
  +nsw 0xFFFFFFBF))
  =>
(zext(x0) << 3)
  + 0xFFFFFFFFFFFFFFBF
```

```
src:
  andi    a0, a0, 255
  slli    a0, a0, 3
  addi    a0, a0, -65
  slli    a0, a0, 32
  srli    a0, a0, 32
tgt:
  andi    a0, a0, 255
  slli    a0, a0, 3
  addi    a0, a0, -65
```

```
(trunc((x0:i80 >>l 63))
 & 1) ^ 1
  =>
zext((
 (x0 & 0x8000000000000000)
  == 0))
```

```
src:
  mov     rax, rdi
  shr     rax, 63
  xor     eax, 1
tgt:
  xor     eax, eax
  bt      rdi, 63
  setae   al
```

```
zext(x:i32) -
  zext((x %u y:i32))
    =>
zext((x - (x %u y)))
```

```
src:
  mov     eax, edi
  mov     ecx, edi
  xor     edx, edx
  div     esi
  sub     rcx, rdx
  mov     rax, rcx
tgt:
  mov     eax, edi
  xor     edx, edx
  div     esi
  sub     edi, edx
  mov     rax, rdi
```

Takeaways

AI for compiler work - Trust, but verify.

Formal methods to verify and fix results.

Scan the QR code to connect with us.
NVIDIA is the engine of the world's AI infrastructure.
We are the world leader in accelerated computing.

Check out our open career opportunities here:
https://www.nvidia.com/en-us/about-nvidia/careers/