LifeJacket: Verifying Precise Floating-Point Optimizations in LLVM

Andres Nötzli, Fraser Brown

Stanford University
Motivating Example

Suppose we want to optimize:

```c
float y = +0.0 - (-x);
```
Motivating Example

Suppose we want to optimize:

```c
float y = +0.0 - (-x);
```

How about:

```c
float y = x;
```
Motivating Example

Suppose we want to optimize:

```c
float y = +0.0 - (-x);
```

How about:

```c
float y = x;
```

Nope: if \( x = -0.0 \) then \( y = +0.0 \)
Motivating Example

So, we know that: \(+0.0\) - \((-x)\) \(\neq x\).

Who cares? \(+0.0\) == \(-0.0\) is true.
Motivating Example

So, we know that: $+0.0 - (-x) \neq x$.

Who cares? $+0.0 == -0.0$ is true.

Well, $\frac{1}{0} = \infty$, $\frac{1}{-0} = -\infty$ and $\infty \neq -\infty$. Ouch.
Motivating Example

So, we know that: \( +0.0 - (-x) \neq x \).

Who cares? \(+0.0 == -0.0\) is true.

Well, \(\frac{1}{0} = \infty\), \(\frac{1}{-0} = -\infty\) and \(\infty \neq -\infty\). Ouch.

What about:

\[ \infty + -\infty = \text{NaN} \]
Motivating Example

So, we know that: \( +0.0 - (-x) \neq x \).

Who cares? \( +0.0 == -0.0 \) is \textbf{true}.

Well, \( \frac{1}{0} = \infty \), \( \frac{1}{-0} = -\infty \) and \( \infty \neq -\infty \). Ouch.

What about:

\[ \infty + -\infty = \text{NaN} \]

\[ x + \text{NaN} = \text{NaN} \]
Motivating Example

So, we know that: \(+0.0 - (-x) \neq x\).

Who cares? \(+0.0 == -0.0\) is true.

Well, \(\frac{1}{0} = \infty\), \(\frac{1}{-0} = -\infty\) and \(\infty \neq -\infty\). Ouch.

What about:

\[ \infty + -\infty = \text{NaN} \]
\[ x + \text{NaN} = \text{NaN} \]
\[ a + (b + c) \neq (a + b) + c \]
Motivating Example

So, we know that: $+0.0 - (-x) \neq x$.

Who cares? $+0.0 == -0.0$ is true.

Well, $\frac{1}{0} = \infty$, $\frac{1}{-0} = -\infty$ and $\infty \neq -\infty$. Ouch.

What about:

$\infty + -\infty = \text{NaN}$

$x + \text{NaN} = \text{NaN}$

$a + (b + c) \neq (a + b) + c$

$a * (b + c) \neq a * b + a * c$
Motivating Example

So, we know that: \(+0.0 - (-x) \neq x\).

Who cares? \(+0.0 == -0.0\) is true.

Well, \(\frac{1}{0} = \infty\), \(\frac{1}{-0} = -\infty\) and \(\infty \neq -\infty\). Ouch.

What about:

\(\infty + -\infty = NaN\)

\(x + NaN = NaN\)

\(a + (b + c) \neq (a + b) + c\)

\(a \times (b + c) \neq a \times b + a \times c\)

Developers have to manually reason about edge cases.
Alive\textsuperscript{1} is a system for verifying peephole optimizations in LLVM. Verification works as follows:

1. User specifies LLVM optimization.
2. Alive translates specification into SMT queries: 
   \[ \texttt{src} \neq \texttt{tgt} \]
3. Alive uses Z3 to solve the SMT queries.

---

\textsuperscript{1}Nuno P Lopes et al. “Provably correct peephole optimizations with Alive”. In: PLDI. 2015.
Alive\textsuperscript{1} is a system for verifying peephole optimizations in LLVM.

Verification works as follows:

1. User specifies LLVM optimization.
2. Alive translates specification into SMT queries:
   \begin{verbatim}
   src \neq tgt
   \end{verbatim}
3. Alive uses Z3 to solve the SMT queries.

Great, but no floating-point arithmetic.

LifeJacket = Alive + Floating-Point Arithmetic

\textsuperscript{1}Nuno P Lopes et al. “Provably correct peephole optimizations with Alive”. In: PLDI. 2015.
Satisfiability Modulo Theory (SMT) solvers

**Input**
First-order logic formula extended with various functions and predicates.

**Example**
\[(x < y) \land (y < x - 1)\]

**Output**
A variable assignment that makes the formula true or unsatisfiable.
### LifeJacket Example

#### (Incorrect) optimization from before

\[ +0.0 - (-x) \Rightarrow x \]

#### In LifeJacket

\[
\begin{align*}
%a &= \text{fsub} \ -0.0, \ %x \\
%r &= \text{fsub} \ +0.0, \ %a \\
\quad &= \Rightarrow \\
%r &= \ %x
\end{align*}
\]

Note: No need to explicitly annotate type width.
(Incorrect) optimization from before

\[ +0.0 - (-x) \Rightarrow x \]

In LifeJacket

\[
\begin{align*}
%a &= fsub -0.0, %x \\
%r &= fsub +0.0, %a \\
&= \Rightarrow \\
%r &= %x
\end{align*}
\]

Note

No need to explicitly annotate type width.
### Input

\[
\begin{align*}
%a &= \text{fsub} \ -0.0, \ %x \\
%r &= \text{fsub} \ +0.0, \ %a \\
&=\ \\
%r &= %x
\end{align*}
\]

### Output

**ERROR:** Mismatch in values of f32 %r  
**Example:**  
%x f32 = -0.0 (0x8000000000000000)  
%a f32 = +0.0 (0x0000000000000000)  
**Source value:** +0.0 (0x0000000000000000)  
**Target value:** -0.0 (0x8000000000000000)
LifeJacket Example

Input

%!a = fsub -0.0, %x
%!r = fsub +0.0, %a

=>
%!r = %x

Output

ERROR: Mismatch in values of f32 %r
Example:
%!x f32 = -0.0 (0x8000000000000000)
%!a f32 = +0.0 (0x0000000000000000)
Source value: +0.0 (0x0000000000000000)
Target value: -0.0 (0x8000000000000000)

Note

Precise optimizations.
Agenda

1. Floating-point types and instructions
2. Fast-math flags

Not today: Floating-point predicates
Agenda

1. Floating-point types and instructions
2. Fast-math flags

Not today: Floating-point predicates
## LifeJacket: Floating-point types and instructions

<table>
<thead>
<tr>
<th>Type support</th>
<th>half, single, double</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary instructions</td>
<td>fadd, fsub, fmul, fdiv, frem</td>
</tr>
<tr>
<td>Conversions</td>
<td>fptrunc, fpext, fptoui, fptosi, uitofp, sitofp</td>
</tr>
<tr>
<td>Other</td>
<td>fabs, fcmp</td>
</tr>
</tbody>
</table>
## LifeJacket: Floating-point types and instructions

<table>
<thead>
<tr>
<th>Type support</th>
<th>half, single, double</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary instructions</td>
<td>fadd, fsub, fmul, fdiv, frem</td>
</tr>
<tr>
<td>Conversions</td>
<td>fptrunc, fpext, fptoui, fptosi, uitofp, sitofp</td>
</tr>
<tr>
<td>Other</td>
<td>fabs, fcmp</td>
</tr>
</tbody>
</table>

Basic operations: direct translation to SMT-LIB operation
1. Floating-point types and instructions
2. Fast-math flags

Not today: Floating-point predicates
LifeJacket supports three fast-math flags on instructions:

- **nnan**: Assume arguments and result are not NaN. Result undefined over NaNs.
- **ninf**: Assume arguments and result are not ±∞. Result undefined over ±∞.
- **nsz**: Allow optimizations to treat the sign of a zero argument or result as insignificant.
Example

\[ x + (0 - x) \Rightarrow 0 \]
Example

\[ x + (0 - x) \Rightarrow 0 \]

In LifeJacket

Precondition: AnyZero(C0)
%a = fsub nnan ninf C0, %x
%r = fadd %x, %a
  =>
%r = 0.0
Example

\[ x + (0 - x) \Rightarrow 0 \]

In LifeJacket

Precondition: AnyZero(C0)

\%a = fsub nnan ninf C0, \%x

\%r = fadd \%x, \%a

\Rightarrow

\%r = 0.0

Translation of \texttt{nnan/ninf}:

If C0 or \%x or \%a is NaN/±∞ then \%a unconstrained
Example

\[ x + (0 - x) \Rightarrow 0 \]

In LifeJacket

**Precondition:** \( \text{AnyZero}(C0) \)
\[ \%a = \text{fsub} \ nnan \ ninf \ C0, \ %x \]
\[ \%r = \text{fadd} \ %x, \ %a \]
\[ => \]
\[ \%r = 0.0 \]

Translation of \( \text{nnan}/\text{ninf} \):

If \( C0 \ or \ %x \ or \ %a \) is \( \text{NaN}/\pm\infty \) then \( %a \) unconstrained

Correct?
LifeJacket: Fast-Math Flags Example

Example

\[ x + (0 - x) \Rightarrow 0 \]

In LifeJacket

**Precondition: AnyZero(C0)**

\[
%a = \text{fsub nnan ninf C0, } %x \\
%r = \text{fadd } %x, %a \\
\Rightarrow \\
%r = 0.0
\]

Translation of `nnan/ninf`:

If \( C0 \) or \( %x \) or \( %a \) is \( \text{NaN} / \pm \infty \) then \( %a \) unconstrained

Correct? No, consider \( %x = \text{NaN} \).
Results
## Results (LLVM 3.7.1)

<table>
<thead>
<tr>
<th>File</th>
<th>Verified</th>
<th>Timeouts</th>
<th>Bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddSub</td>
<td>7</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MulDivRem</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Compares</td>
<td>11</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Simplify</td>
<td>22</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>43</strong></td>
<td><strong>3</strong></td>
<td><strong>8</strong></td>
</tr>
</tbody>
</table>
## Results (LLVM 3.7.1)

<table>
<thead>
<tr>
<th>File</th>
<th>Verified</th>
<th>Timeouts</th>
<th>Bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddSub</td>
<td>7</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MulDivRem</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Compares</td>
<td>11</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Simplify</td>
<td>22</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>43</strong></td>
<td><strong>3</strong></td>
<td><strong>8</strong></td>
</tr>
</tbody>
</table>

### Insights

Few timeouts, high bug rate, bugs related to floating-point properties.
LifeJacket currently has some limitations:

- No support for types wider than 64-bit.
- Fixed rounding mode.
- No support for floating-point exceptions/debug information in NaNs.
LifeJacket currently has some limitations:

- No support for types wider than 64-bit.
- Fixed rounding mode.
- No support for floating-point exceptions/debug information in NaNs.

But: Errors found are true errors.
Conclusion

Automatic verification is possible: 43 verified optimizations.
Automatic verification is necessary: 8 bugs.

More automation = More optimizations, more boring compilers.

Would you like to know more?

🔗 https://github.com/4tXJ7f/alive
✉️ noetzli@stanford.edu