Automated Translation Validation for an LLVM Backend

Nader Boushehrinejad Moradi, Ryan Berger, Stefan Mada, John Regehr

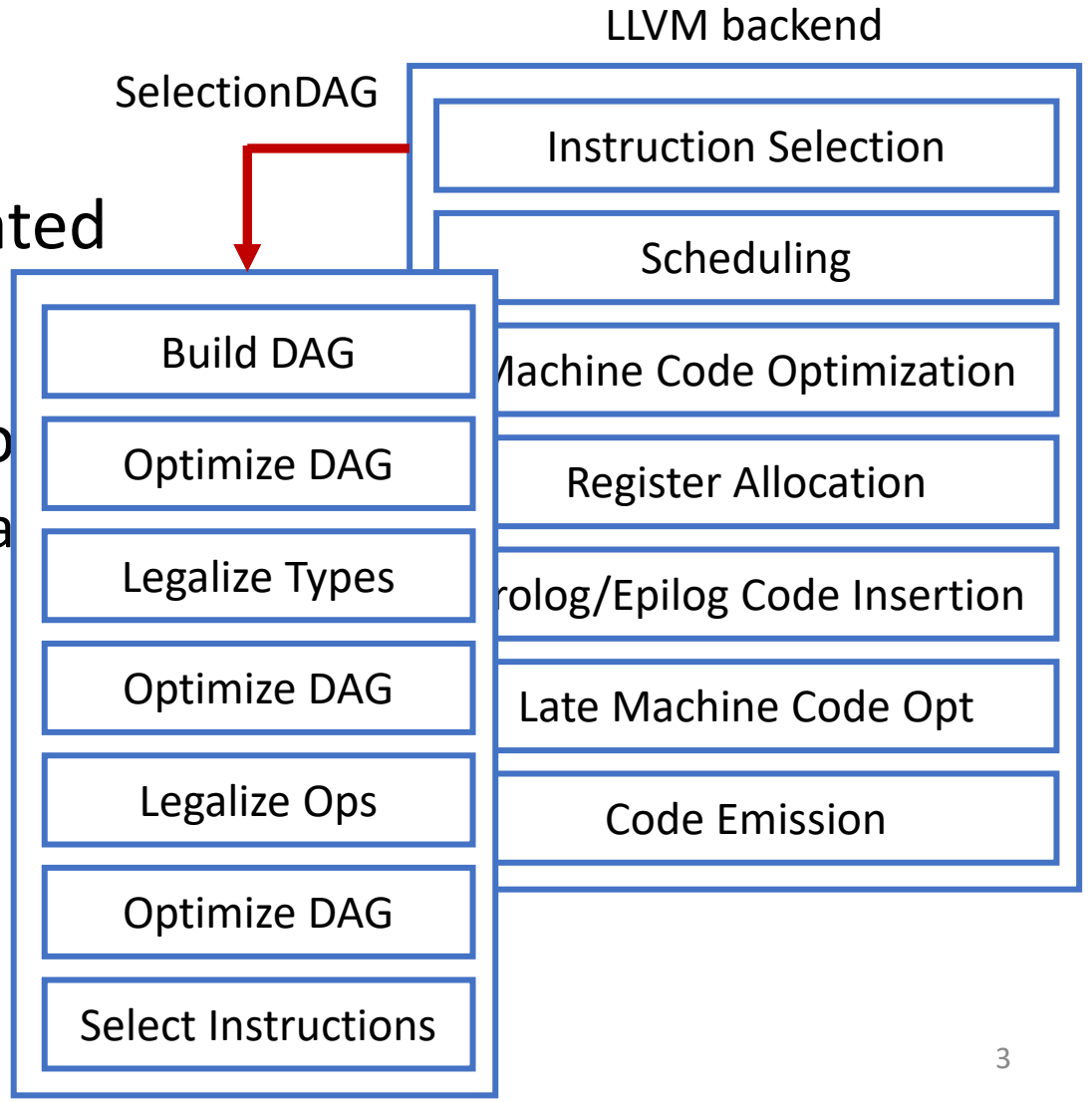
University of Utah

Topic of This Talk

- LLVM supports a variety of backends

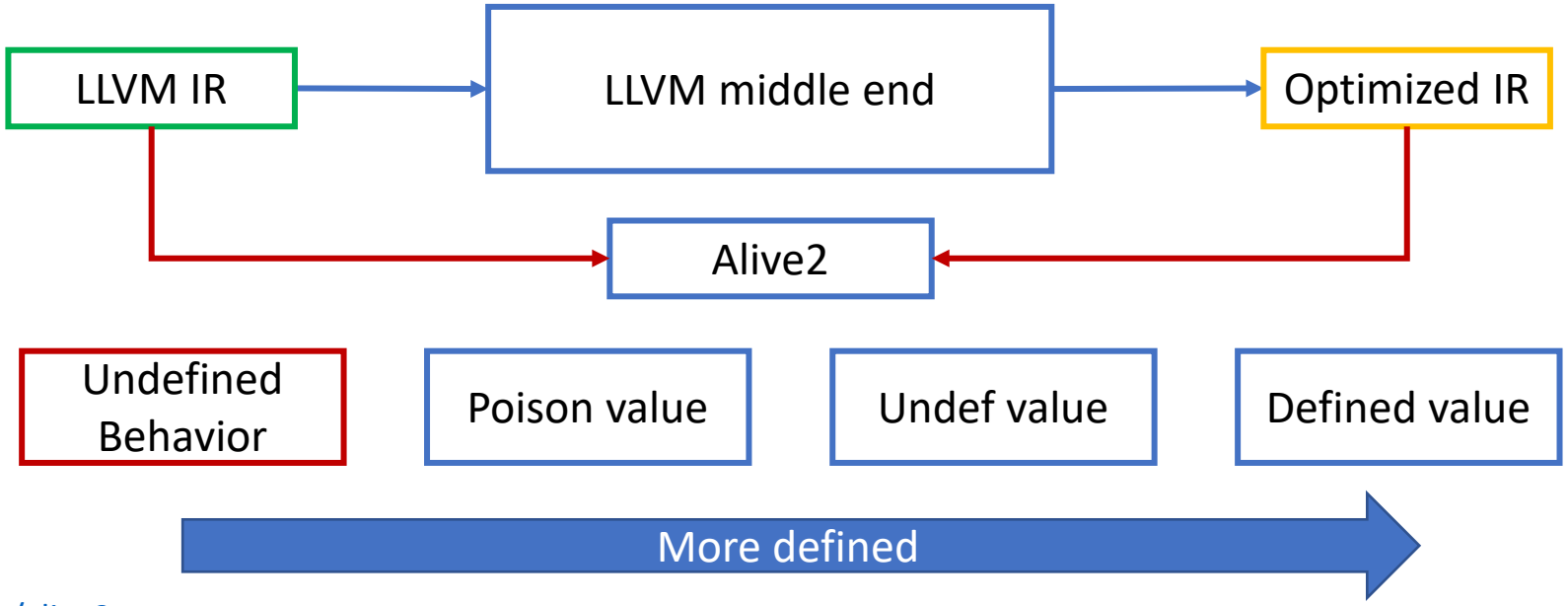
Topic of This Talk

- LLVM supports a variety of backends
- Backend code generation is complicated
- Complexity may result in bugs
- Find bugs using automated translatio
  - Promising results in testing AARch64 ba
  - ARM-TV prototype built using Alive2



Translation Validation in Alive2

- Check for refinement
  - Fully automated
  - No false positives
- Result is either correct, not correct (with example), or a timeout



https://alive2.llvm.org/ce/

https://github.com/AliveToolkit/alive2

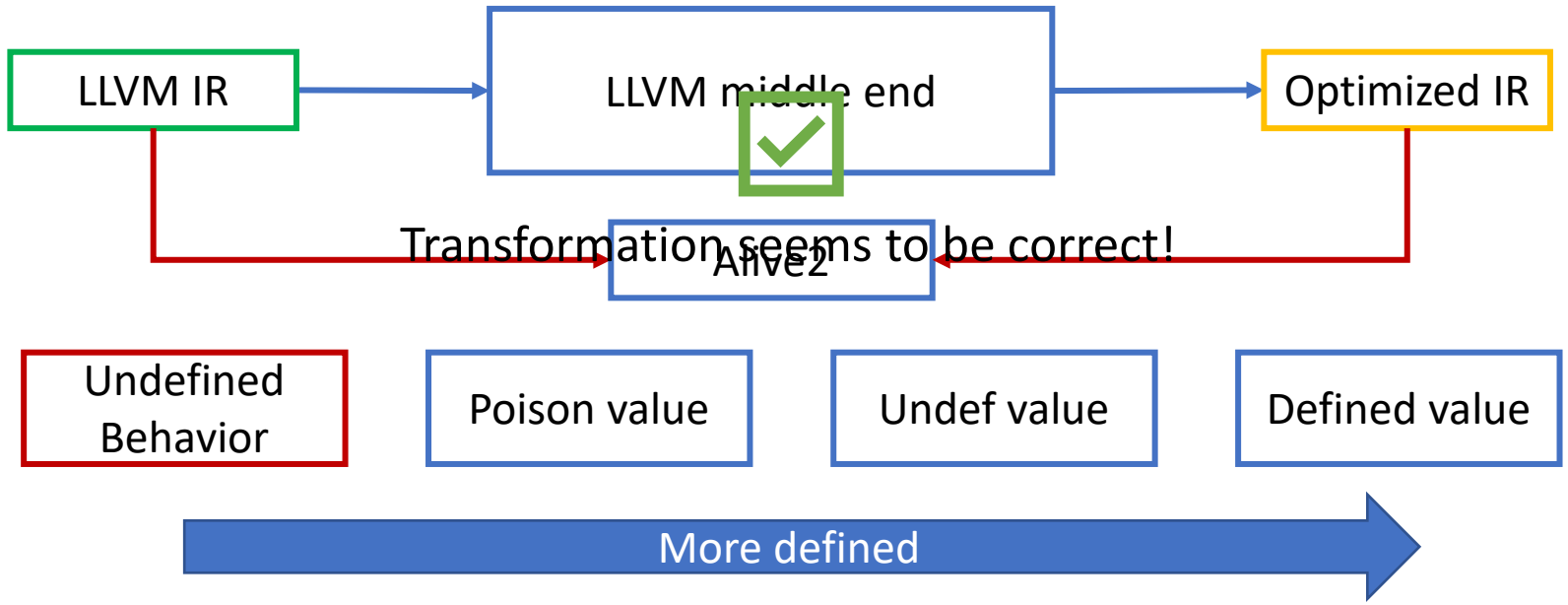
Translation Validation in Alive2

- Result is either correct, not correct (with example), or a timeout

```
1 define i64 @src(i64 %a, i64 %b) {
2  %res = add nuw i64 %a, %b
3  ret i64 %res
4 }
```

→

```
1 define i64 @tgt(i64 %a, i64 %b) {
2  %res = add i64 %a, %b
3  ret i64 %res
4 }
```



Translation Validation in Alive2

- Result is either correct, not correct (with example), or a timeout

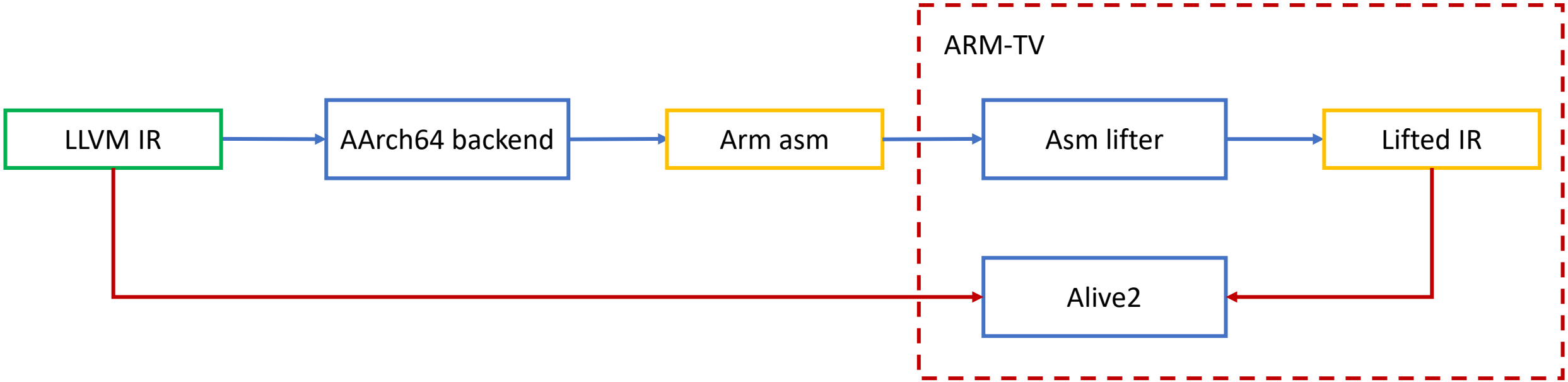
```
1 define i64 @src(i64 %a, i64 %b) {
2  %res = add i64 %a, %b
3  ret i64 %res
4 }
```

```
1 define i64 @tgt(i64 %a, i64 %b) {
2  %res = add nuw i64 %a, %b
3  ret i64 %res
4 }
```

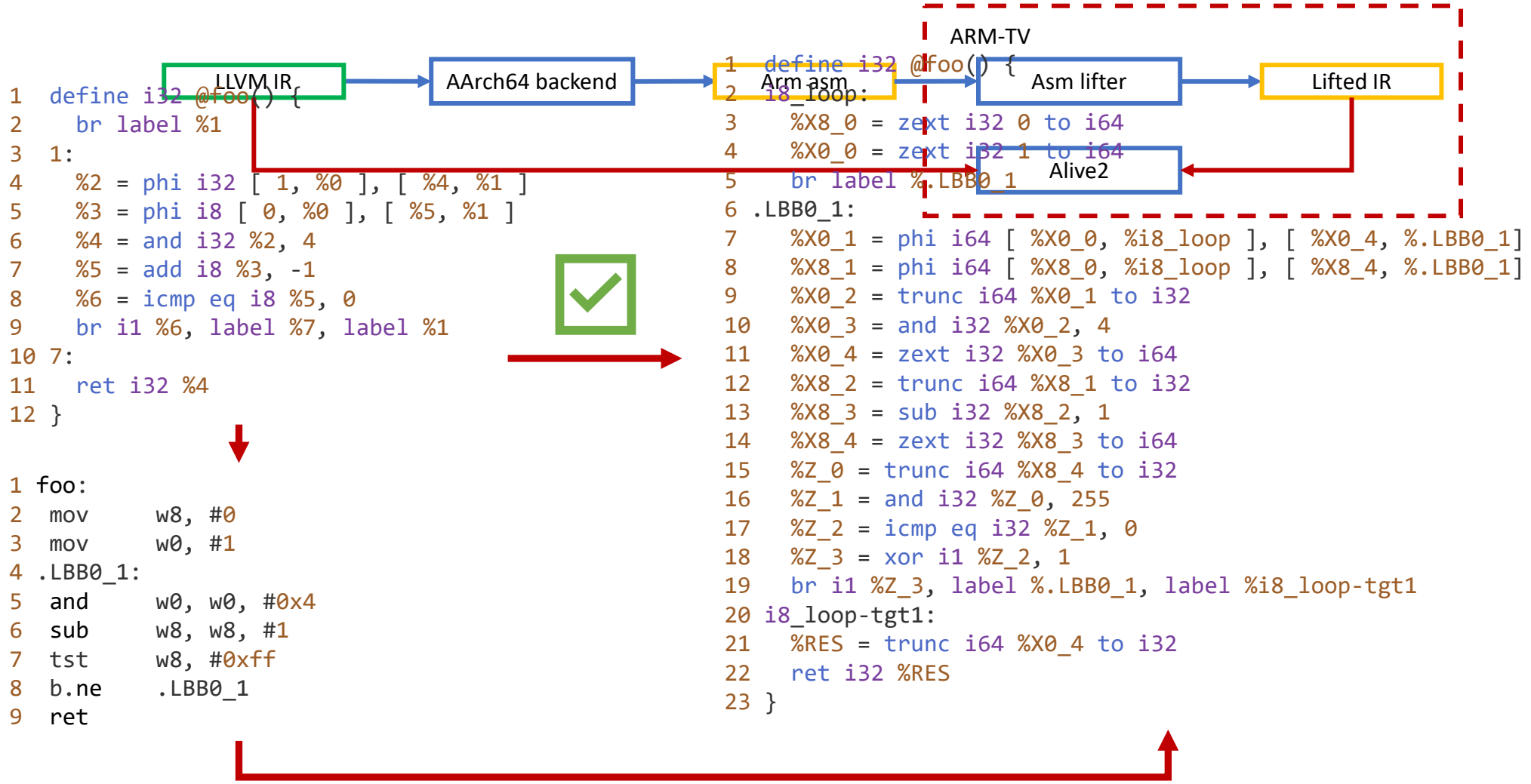
```
Transformation doesn't verify!
ERROR: Target is more poisonous than source

Example:
i64 %a = #x8000000000000000 (9223372036854775808, -9223372036854775808)
i64 %b = #x8000000000000001 (9223372036854775809, -9223372036854775807)

Source:
i64 %res = #x0000000000000001 (1)
Target:
i64 %res = poison
Source value: #x0000000000000001 (1)
Target value: poison
```

Testing the AArch64 Backend with ARM-TV

- Enhance Alive2 to support the AArch64 llvm backend



Testing the AArch64 Backend with ARM-TV



Example

```
1 define i64 @f(i64 %0) {
2  %c = icmp ult i64 %0, 777
3  br i1 %c, label %t, label %f
4 t:
5  %a = add i64 %0, 1
6  ret i64 %a
7 f:
8  ret i64 %0
9 }
```

lifter

```
1 define i64 @f(i64 %X0) {
2  %X0_1 = freeze i64 %X0
3  %PS_0 = icmp uge i64 %X0_1, 777
4  %X0_2 = add i64 %X0_1, 1
5  %X0_3 = select i1 %PS_0, i64 %X0_1, i64 %X0_2
6  ret i64 %X0_3
7 }
```

AArch64 Backend

```
1 f:
2  cmp     x0, #777
3  cinc    x0, x0, lo
4  ret
```

- Model calling convention
- Generate SSA
- Remove poison/undef input
- Update processor state

Example with Different Bitwidths

```
1 define i23 @f23(i23 %0) {
2  %c = icmp ult i23 %0, 777
3  br i1 %c, label %t, label %f
4 t:
5  %a = add i23 %0, 1
6  ret i23 %a
7 f:
8  ret i23 %0
9 }
```

OPT –O2

```
1 define i23 @f(i23 %0) {
2  %c = icmp ult i23 %0, 777
3  %a = zext i1 %c to i23
4  %common.ret.op = add i23 %a, %0
5  ret i23 %common.ret.op
6 }
```

```
1 define i64 @f64(i64 %0) {
2  %c = icmp ult i64 %0, 777
3  br i1 %c, label %t, label %f
4 t:
5  %a = add i64 %0, 1
6  ret i64 %a
7 f:
8  ret i64 %0
9 }
```

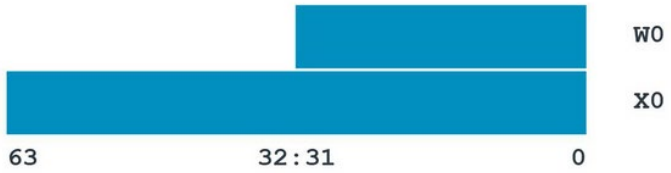
```
1 define i64 @f64(i64 %0) {
2  %c = icmp ult i64 %0, 777
3  %a = zext i1 %c to i64
4  %common.ret.op = add i64 %a, %0
5  ret i64 %common.ret.op
6 }
```

Example with Different Bitwidths

```
1 define i23 @f23(i23 %0) {
2  %c = icmp ult i23 %0, 777
3  br i1 %c, label %t, label %f
4 t:
5  %a = add i23 %0, 1
6  ret i23 %a
7 f:
8  ret i23 %0
9 }
```

AArch64 Backend

```
1 f23:
2  and     w0, #0x7fffff
3  cmp     w0, #777
4  cinc    w0, w0, lo
5  ret
```




```
1 define i64 @f64(i64 %0) {
2  %c = icmp ult i64 %0, 777
3  br i1 %c, label %t, label %f
4 t:
5  %a = add i64 %0, 1
6  ret i64 %a
7 f:
8  ret i64 %0
9 }
```

```
1 f64:
2  cmp     x0, #777
3  cinc    x0, x0, lo
4  ret
```

- HW registers have a specific bitwidth

Example with Different Bitwidths

```
1 define i23 @f23(i64 %0) {
2  %0_t = trunc i64 %0 to i23
3  %c = icmp ult i23 %0_t, 777
4  br i1 %c, label %t, label %f
5 t:
6  %a = add i23 %0_t, 1
7  ret i23 %a
8 f:
9  ret i23 %0_t
10 }
```

```
1 f23:
2  and     w0, #0x7fffff
3  cmp     w0, #777
4  cinc    w0, w0, lo
5  ret
```

lifter

```
1 define i23 @f23(i64 %X0) {
2 f:
3   %X0_1 = freeze i64 %X0
4   %X0_2 = trunc i64 %X0_1 to i23
5   %X0_3 = zext i23 %X0_2 to i32
6   %X0_4 = zext i32 %X0_3 to i64
7   %X0_5 = trunc i64 %X0_4 to i32
8   %X0_6 = and i32 %X0_5, 8388607
9   %Z_0 = icmp uge i32 %X0_6, 777
10  %X0_7 = trunc i64 %X0_4 to i32
11  %X0_8 = trunc i64 %X0_4 to i32
12  %X0_9 = add i32 %X0_8, 1
13  %RES = select i1 %Z_0, i32 %X0_7, i32 %X0_9
14  %RES_T = trunc i32 %RES to i23
15  ret i23 %RES_T
16 }
```

- HW registers have a specific bitwidth

Modeling LLVM's Signext/Zeroext Attributes

- zeroext

  This indicates to the code generator that the parameter or return value should be zero-extended to the extent required by the target's ABI by the caller (for a parameter) or the callee (for a return value).

- signext

  This indicates to the code generator that the parameter or return value should be sign-extended to the extent required by the target's ABI (which is usually 32-bits) by the caller (for a parameter) or the callee (for a return value).

Modeling LLVM's Signext/Zeroext

- signext

  This indicates to the code generator that the parameter or return value should be sign-extended to the extent required by the target's ABI (which is usually 32-bits) by the caller (for a parameter) or the callee (for a return value).

```
1 define i8 @not_i8(i8 %0) {
2  %2 = xor i8 %0, 255
3  ret i8 %2
4 }
```

```
1 not_i8:
2  mvn     w0, w0
3  ret
```

```
1 define signext i8 @not_i8_s(i8 %0) {
2  %2 = xor i8 %0, 255
3  ret i8 %2
4 }
```

```
1 not_i8_s:
2  mvn     w8, w0
3  sxtb    w0, w8
4  ret
```

```
1 define signext i8 @not_i8_ss(i8 signext %0) {
2  %2 = xor i8 %0, 255
3  ret i8 %2
4 }
```

```
1 not_i8_ss:
2  mvn     w0, w0
3  ret
```

- signext

  This indicates to the code generator that the parameter or return value should be sign-extended to the extent required by the target's ABI (which is usually 32-bits) by the caller (for a parameter) or the callee (for a return value).

```
1 define i8 @not_i8(i8 %0) {
2  %2 = xor i8 %0, 255
3  ret i8 %2
4 }
```

```
1 define i8 @not_i8(i64 %0) {
2  %0_t = trunc i64 %0 to i8
3  %2 = xor i8 %0_t, 255
4  ret i8 %2
5 }
```

```
1 not_i8:
2  mvn     w0, w0
3  ret
```

```
1 define i8 @not_i8(i64 %X0) {
2   %X0_1 = freeze i64 %X0
3   %X0_2 = trunc i64 %X0_1 to i8
4   %X0_3 = zext i8 %X0_2 to i32
5   %X0_4 = xor i32 %X0_3, -1
6   %RES = trunc i32 %X0_4 to i8
7   ret i8 %RES
8 }
```

```
1 define signext i8 @not_i8_s(i8 %0) {
2  %2 = xor i8 %0, 255
3  ret i8 %2
4 }
```

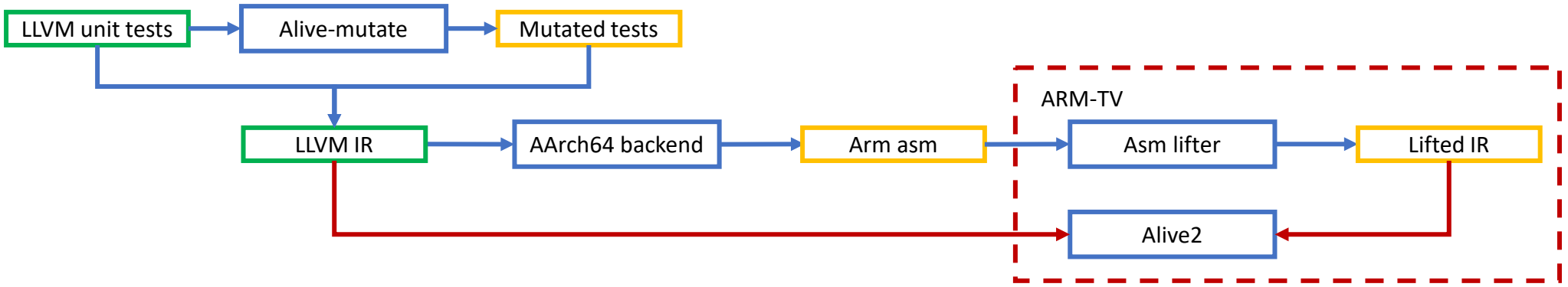
```
1 define signext i64 @not_i8_s(i64 %0) {
2  %0_t = trunc i64 %0 to i8
3  %2 = xor i8 %0_t, 255
4  %2_s = sext i8 %2 to i32
5  %2_z = zext i32 %2_z to i64
6  ret i64 %2_z
7 }
```

```
1 not_i8_s:
2  mvn     w8, w0
3  sxtb    w0, w8
4  ret
```

```
1 define i64 @not_i8_s(i64 %X0) {
2   %X0_1 = freeze i64 %X0
3   %X0_2 = trunc i64 %X0_1 to i8
4   %X0_3 = zext i8 %X0_2 to i32
5   %X0_4 = xor i32 %X0_3, -1
6   %X0_5 = trunc i32 %X0_4 to i8
7   %X0_6 = sext i8 %X0_5 to i32
8   %RES = zext i32 %X0_6 to i64
9   ret i64 %RES
10}
```

Test Case Generation

- OPT-fuzz
- LLVM unit test suite
- Alive-mutate



[Alive-mutate: a fuzzer that cooperates with Alive2 to find LLVM bugs]

Results

- 19 bugs reported

```
1. GlobalIsel miscompiles an llvm.fshl instruction
(https://github.com/llvm/llvm-project/issues/55003)
2. GlobalIsel miscompiles a zero-extended logical shift right
(https://github.com/llvm/llvm-project/issues/55129)
3. Incorrect optimization of sitofp/fptosi roundtrip
(https://github.com/llvm/llvm-project/issues/55150)
4. Miscompilation on a shift followed by an icmp instruction
(https://github.com/llvm/llvm-project/issues/55178)
5. Miscompilation when backend attempts to lower to a rotate instruction
(https://github.com/llvm/llvm-project/issues/55201)
6. Miscompilation with urem and undef
(https://github.com/llvm/llvm-project/issues/55271)
…
```

Bug with Constant Parameters

```
1 define i32 @f() {
2  %1 = sub i8 -66, 0
3  %2 = icmp ugt i8 -31, %1
4  %3 = select i1 %2, i32 1, i32 0
5  ret i32 %3
6 }
```

Simplify

```
1 define i32 @f() {
2  %1 = icmp ugt i8 225, 190
3  %2 = select i1 %1, i32 1, i32 0
4  ret i32 %2
5 }
```

Bug with Constant Parameters

```
1 define i32 @f() {
2  %1 = sub i8 -66, 0
3  %2 = icmp ugt i8 -31, %1
4  %3 = select i1 %2, i32 1, i32 0
5  ret i32 %3
6 }
```

Simplify

```
1 define i32 @f() {
2   ret i32 1
3 }
```

llc 14.0.0

```
1 f:
2   mov     w0, wzr
3   ret
```

llc (trunk)

```
1 f:
2   mov     w0, #1
3   ret
```

https://github.com/llvm/llvm-project/issues/55342

Bug Involving Signext Attribute

signext

This indicates to the code generator that the parameter or return value should be sign-extended to the extent required by the target's ABI (which is usually 32-bits) by the caller (for a parameter) or the callee (for a return value).

arm Table 3, Mapping of C & C++ built-in data types

C/C++ Type	Machine Type	Notes
_Bool/bool	unsigned byte	C99/C++ only. False has value 0 and True has value 1

For an i1, the caller implicitly zero extends to an i8 first, then it sign extends

Bug Involving Signext Attribute

For an i1, the caller implicitly zero extends to an i8 first, then it sign extends

```
1 define i32 @sub_1(i1 signext %w0, i32 %w1) {
2  %w0_z = zext i1 %w0 to i32
3  %res = sub i32 %w1, %w0_z
4  ret i32 %res
5 }
```

```
1 sub_1:
2  add     w0, w1, w0
3  ret
```

%w0 = #x1 %w1=#x00000000
%w0_z = #x00000001
%res = #x11111111

%w0 = #x0000001 %w1=#x00000000
%res = #x0000001

https://github.com/llvm/llvm-project/issues/57181

Bug Involving Signext Attribute

With a signext attribute, the sign extension takes precedence over the implicit zero extension

```
1 define i32 @sub_1(i1 signext %w0, i32 %w1) {
2  %w0_z = zext i1 %w0 to i32
3  %res = sub i32 %w1, %w0_z
4  ret i32 %res
5 }
```

%w0 = #x1 %w1=#x00000000
%w0_z = #x00000001
%res = #x11111111

```
1 sub_1:
2  add     w0, w1, w0
3  ret
```

%w0 = #x1111111 %w1=#x00000000
%res = #x1111111

https://github.com/llvm/llvm-project/issues/57181

Ongoing Work

- Memory support
  - Pointer types
  - Function calls
- Vector instructions
  - Derive lifting code using formal semantics
- Support other backends
  - RISC-V

Thank you

ARM-TV

TODO

Alive2

https://github.com/nbushehri/alive2/tree/arm-tv

https://github.com/AliveToolkit/alive2

https://alive2.llvm.org/ce/