Undef and Poison: Present and Future

Juneyoung Lee
Seoul National University

This talk is based on joint work with
Sanjoy Das, Chung-Kil Hur, Nuno P. Lopes, David Majnemer, John Regehr

What is This Talk About?

- LLVM has a notion of undef & poison values.
- Their semantics has been unclear, causing real-world problems.

[llvm-dev] A bug related with undef value when bootstrap MemorySSA.cpp

Mon Jul 17 01:24:19 PDT 2017

```
Every transformation above seems of no problem, but the composition
result is wrong. It is still not clear which transformation to blame.
```

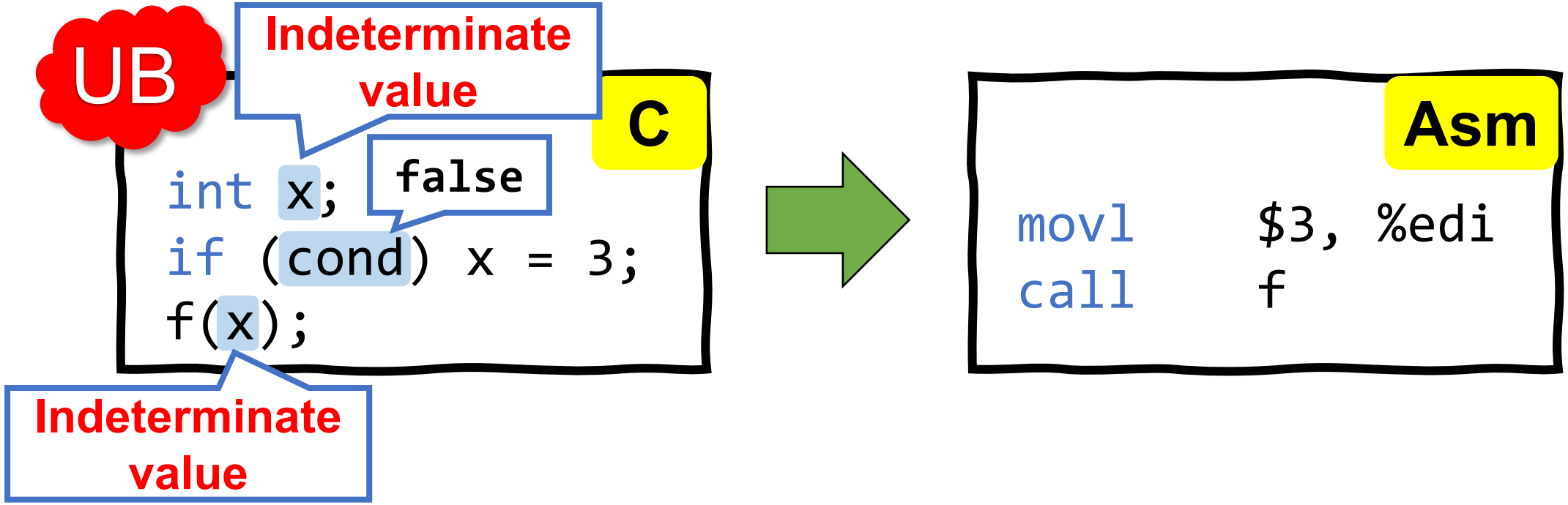
- Recently, efforts have been made to address the problem.
- I will talk about the background, current status, and future directions.

Background

Undefined Behavior,
Undef, and Poison

Undefined Behavior

- Behavior of a program that violates the language standard
- **Behavioral refinement:** Compiler assumes the source has no UB



Motivation for Undef

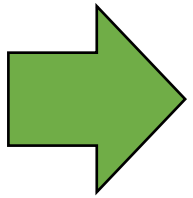
Problem

IR didn't have a notion of 'uninitialized value'

C

```
int x;
if (cond) x = 3;

f(x);
```



IR

```
; br cond, ...
x = phi(3, undef)
call f(x)
```

New LLVM 'undef' Value, http://www.nondot.org/sabre/LLVMNotes/UndefinedValue.txt

Undef ≠ Indeterminate Value

- **Example: C's bitfield**

```
struct {
  int x: 2, y: 6;
} a;
a.x = 1;
```

UB

Indeterminate value

```
a = alloca
b = load a
v = (b & ~3) | 1
store v, a
```

IR

New LLVM 'undef' Value, http://www.nondot.org/sabre/LLVMNotes/UndefinedValue.txt

Definition of Undef

- `undef` of type T is the set consisting of all defined values of T.
- A (partially) undefined value is a subset of `undef`.
- An operation on undefined values is defined in element-wise manner

C

```
struct {
  int x: 2,
} a;
a.x = 1;
```

undef = {0, 1, ..., 255}
8 bits: ********

IR

```
a = alloca i8
b = load i8 a
v = (b & ~3) | 1
store v, a
```

********01**

********00**

Motivation for Poison

Problem

Needed a value that represents signed overflow in LLVM IR,

But `undef` was too weak & UB was too strong.

- **Example: Widening an induction variable**

IR

```
int32_t i = 0;
while (i <= y) {
  arr[i] = ...;
  i = i
}
```

Needs to signext i to 64 (expensive)

IR

```
int64_t i = 0;
while (i <= y) {
  arr[i] = ...;
  i = i
}
```

No signext needed (cheap)

The nsw story, https://groups.goo[illegible]F-g/m/5Ektu6vM_0oJ

Motivation for Poison

Problem

Needed a value that represents signed overflow in LLVM IR,

But `undef` was too weak & UB was too strong.

- **Example: Widening an induction variable**

```
int32_t i = 0;
while (i <= y) {
  arr[i] = ...
  i = i + 1;
}
```

always `true`

IR

INT32_MAX

```
int64_t i = 0;
while (i <= y) {
  arr[i] = ...
  i = i + 1;
}
```

can be `false`

IR

INT32_MAX

The nsw story, https://groups.google.com/g/llvm-dev/c/sDYaYV_ZF-g/m/5Ektu6vM_0oJ

Motivation for Poison

Problem

Needed a value that represents signed overflow in LLVM IR,

But `undef` was too weak & UB was too strong.

- **Example: Widening an induction variable**

```
int32_t i = 0;
while (i <= y) {
  arr[i] = ...;
  i = i +nsw 1;
}
```

IR

always true

INT32_MAX

INT32_MAX

poison!

```
int64_t i = 0;
while (i <= y) {
  arr[i] = ...;
  i = i +nsw 1;
```

IR

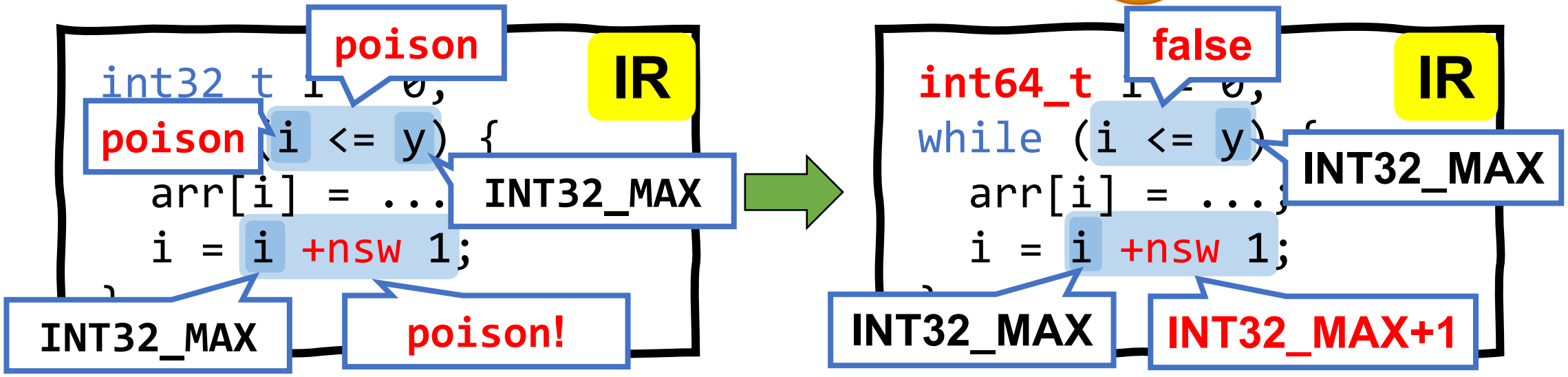
1. `undef` ≤ `INT32_MAX` is still true
2. Raising UB blocks code motion

The nsw story, https://groups.google.com/g/llvm-dev/c/sDYaYV_ZF-g/m/5Ektu6vM_0oJ

Definition of Poison

- `poison` is a special value that represents a violation of an assumption
- Each operation either propagates `poison` or raise UB
- (Property) `poison` is refined by any (defined or undefined) value

`poison` → false is allowed



Comparison of Undef and Poison

1. `poison` and `undef` can fold to a different (defined) value at each use

undef

```
y = load uninit_var
use1(y)
use2(y)
```

→

```
use1(0)
use2(1)
```

poison **poison**

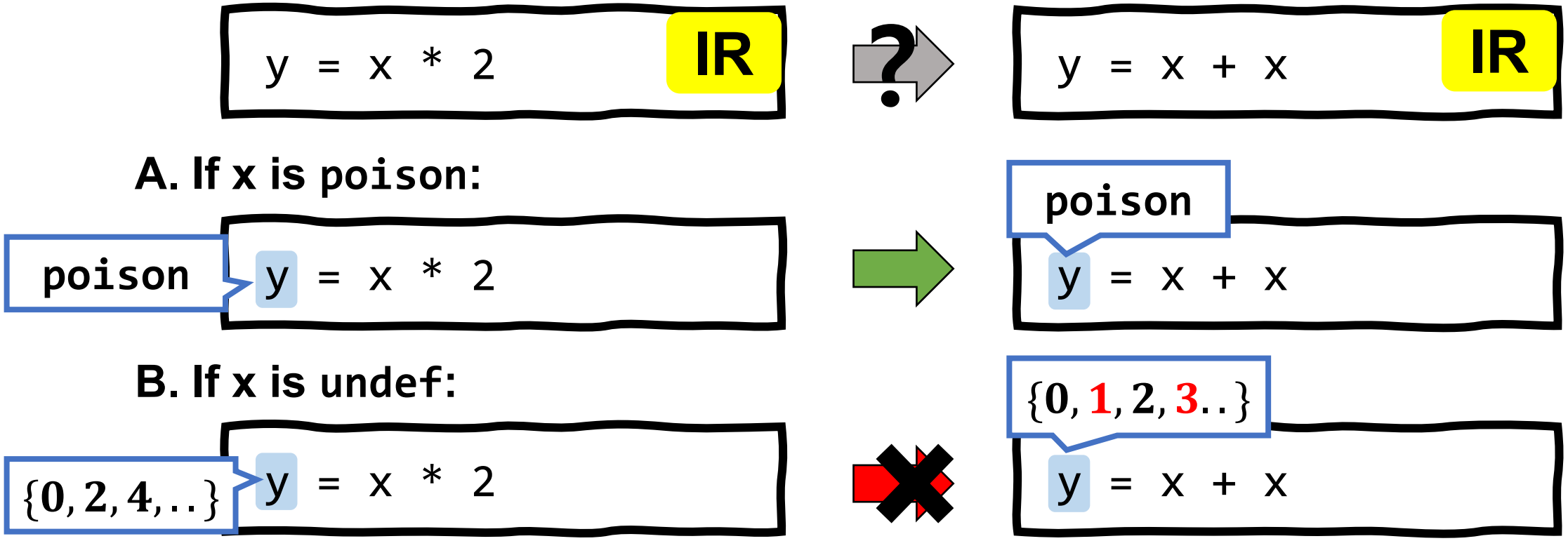
```
z = INT_MAX < (INT_MAX +nsw 1)
use1(z)
use2(z)
```

→

```
use1(0)
use2(1)
```

Comparison of Undef and Poison

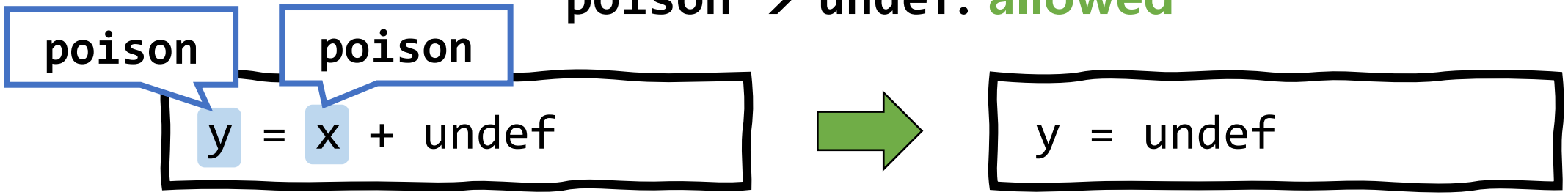
2. Undefined values do not admit certain arithmetic properties



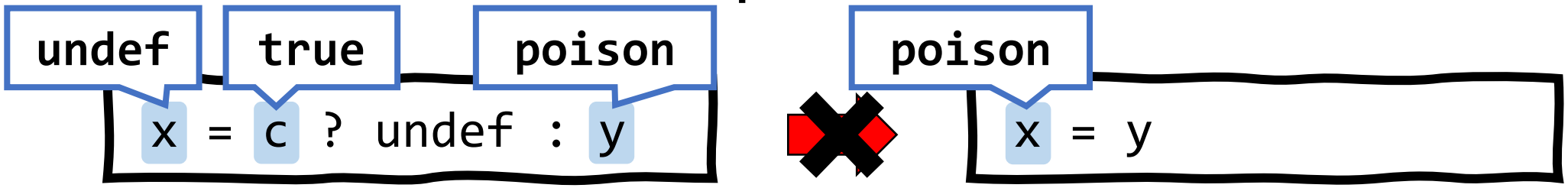
Comparison of Undef and Poison

3. `poison` is more undefined than `undef`

poison → undef: allowed



undef → poison: disallowed



https://reviews.llvm.org/D83360

Comparison of Undef and Poison

4. `poison` cannot be used for uninitialized bitfields

C

```
struct {
  int x: 2, y: 6;
} a;
a.x = 1;
```

poison

IR

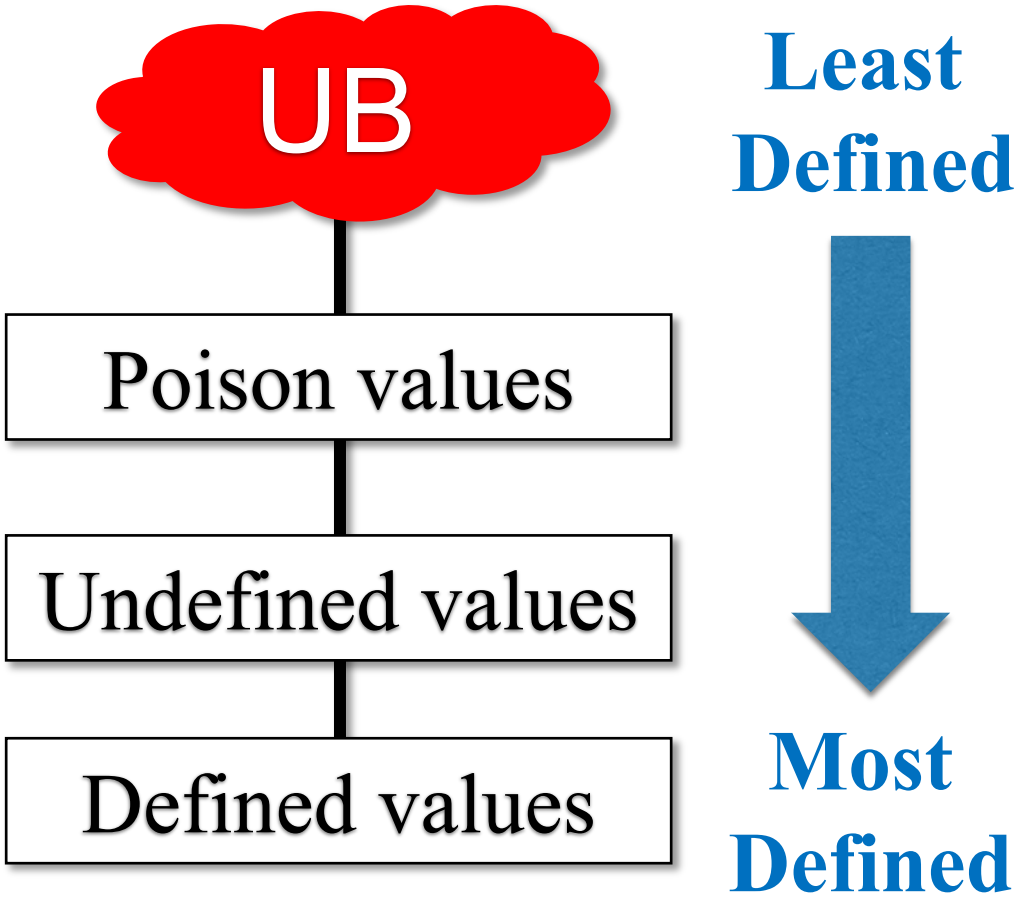
```
a = alloca i8
b = load i8 a
v = (b & ~3) | 1
store v, a
```

poison

poison

Summary: UB, Undef, and Poison

- Undefined behavior is the strongest one
- `poison` is a notion of deferred UB
- Undefined values are sets of values



Recent Progresses in Fixing UB-related Problems in LLVM

1. Semantics Are Clarified at LangRef.

Branch

```
br undef, A, B
```

```
switch undef, ...
```

UB

```
// MSAN does not like undefs as branch condition which can be introduced
// with "explicit branch".
if (ExtraCase && BB->getParent()->hasFnAttribute(Attribute::SanitizeMemory))
  return false;
```

Ternary Op.

```
z = select poison, x, y
```

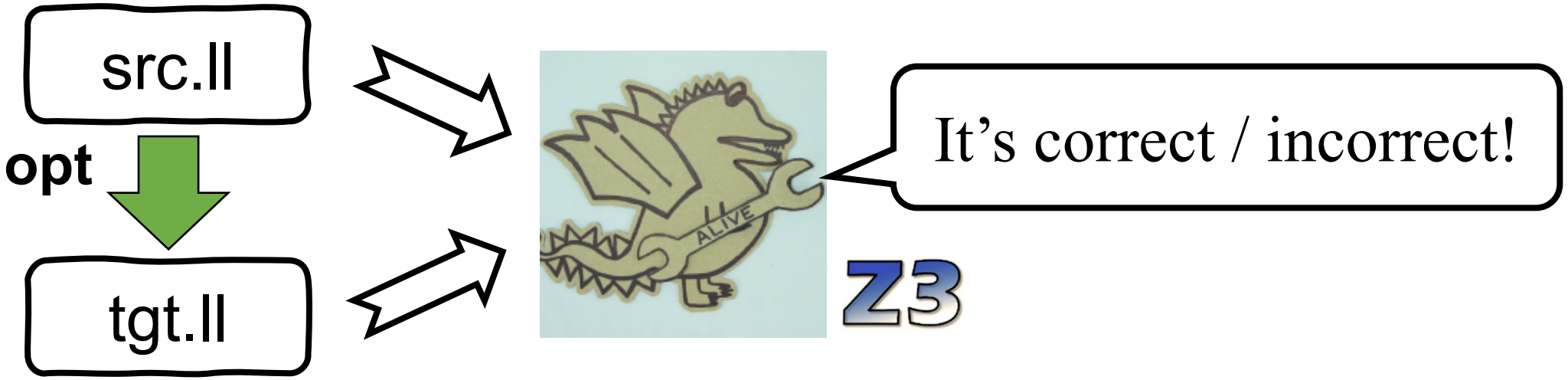
$z = \mathbf{poison}$

And also shufflevector's undef mask, memset(undef, val, 0), padding of aggregates, ...

https://reviews.llvm.org/D76973 https://reviews.llvm.org/D86189
https://reviews.llvm.org/D70641 https://reviews.llvm.org/D86643

2. Undef/Poison-related Bugs Are Found with Alive2

- Alive2 is a translation validation tool for LLVM: https://alive2.llvm.org



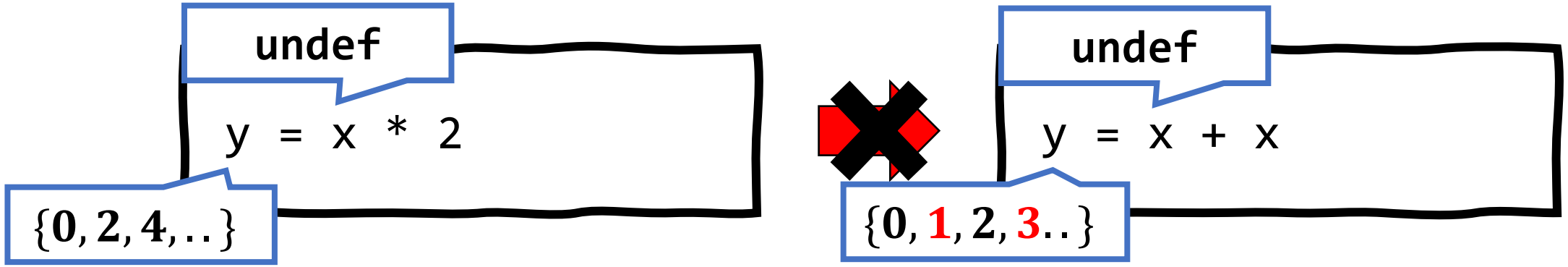
- llvm/test/Transforms: 23 bugs reported, 17 fixed, 37 failures remaining
- Project Zero LLVM Bugs: https://web.ist.utl.pt/nuno.lopes/alive2/

3. Freeze to the Rescue

- Officially added to LLVM 10.0

Definition of "`y = freeze x`"

- If x is `poison` or undefined value: return a defined, nondeterministically chosen, value
- Otherwise: return `x`

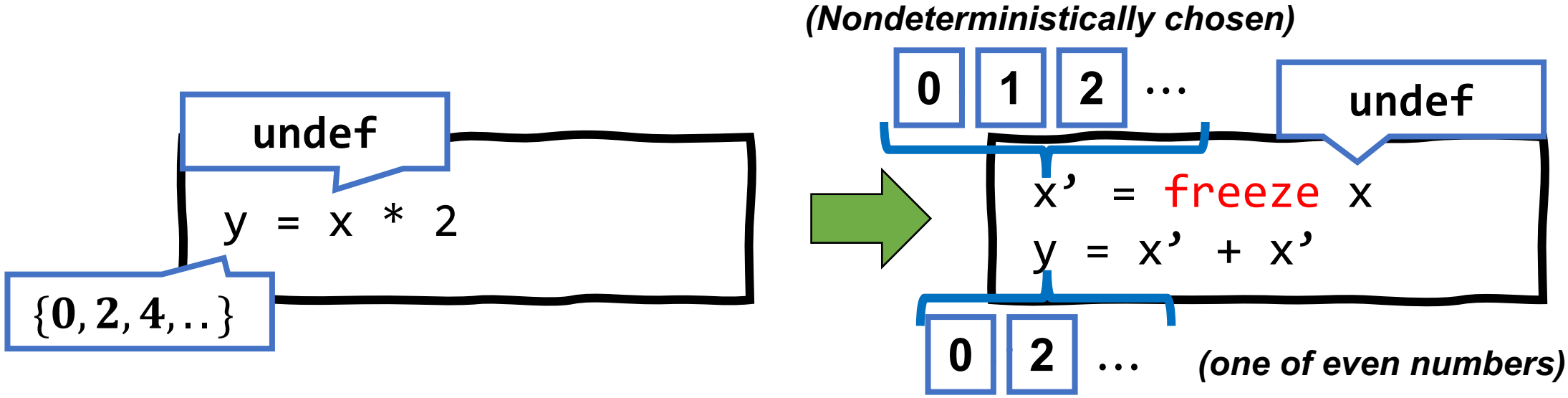


3. Freeze to the Rescue

- Officially added to LLVM 10.0

Definition of "`y = freeze x`"

- If x is `poison` or undefined value: return a defined, nondeterministically chosen, value
- Otherwise: return x



3. Freeze to the Rescue

Fixing "Select → Branch" Using Freeze

```
z = select c, x, y
```

poison

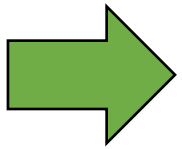
poison

UB

poison

```
if (c) z = x
else   z = y
```

```
z = select c, x, y
```



true

false

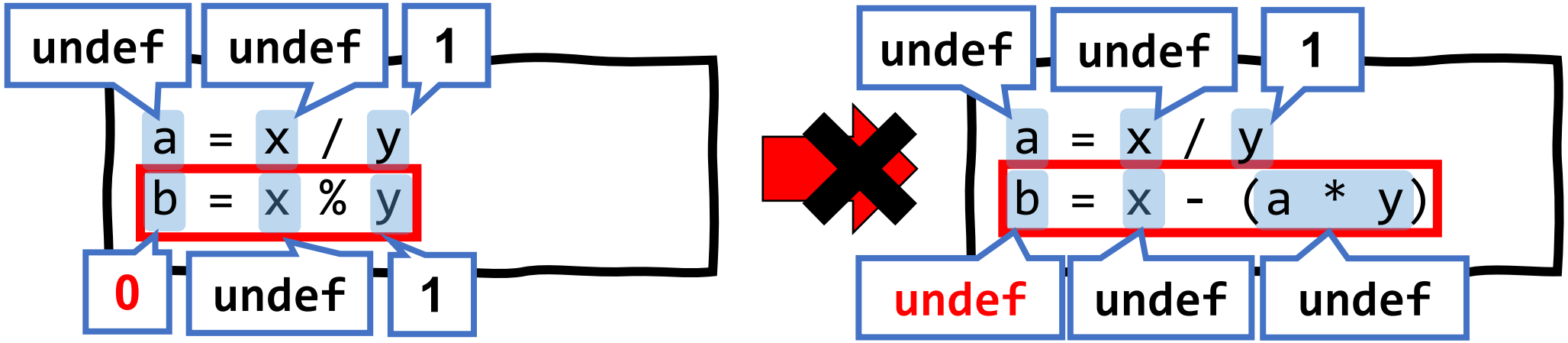
poison

```
if (freeze(c)) z = x
else   z = y
```

https://reviews.llvm.org/D84940
https://reviews.llvm.org/D76179

3. Freeze to the Rescue

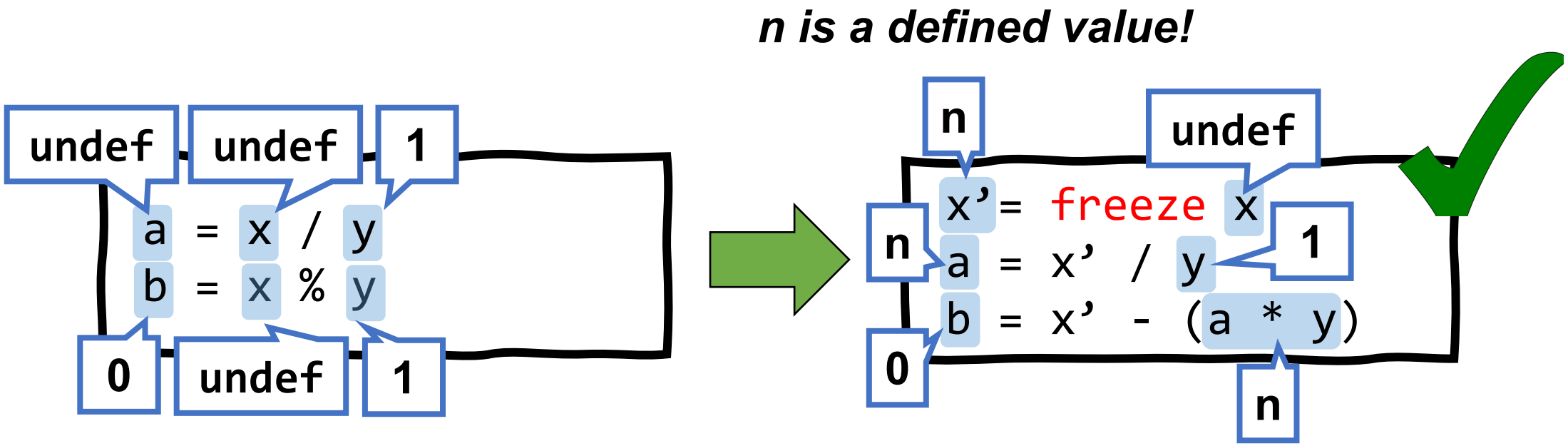
Fixing DivRemPairs Using Freeze



https://reviews.llvm.org/D76483

3. Freeze to the Rescue

Fixing DivRemPairs Using Freeze



In the full patch, y is frozen as well because giving an undefined value to y causes a bug too.

Performance Regression Matters

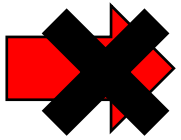
- There are optimizations/analyses unaware of freeze
- Fixing DivRemPairs: ~2% slowdown in 505.mcf_r with LTO, -O3
  - Reason: SCEV wasn't aware of freeze → LSR disabled
  - Solution: added a pass that hoists freeze out of a loop to remove the slowdown

https://reviews.llvm.org/D77523

4. Some Optimizations Were Removed

Folding select with undef operand

```
x = c ? undef : y
```



```
x = y
```

- It can be easily fixed with freeze, but simply disabled

https://reviews.llvm.org/D83360
https://reviews.llvm.org/D85684

5. Patches Have Landed to Recover Performance

A. Insert fewer freeze instructions

- `ValueTracking::isGuaranteedNotToBeUndefOrPoison`
- Library functions (e.g. `printf`) have `noundef` at arguments/return values

B. Make optimizations & analyses aware of freeze

- GVN, LICM, EarlyCSE, JumpThreading, ... are aware of freeze
- `computeKnownBits`, `isKnownZero` understand freeze

https://reviews.llvm.org/D29013
https://reviews.llvm.org/D75808
https://reviews.llvm.org/D85345

Future Directions

1. Use Non-Undef/Poison Assumption From Source Language

- (Ongoing) Attach `noundef` to function arguments when lowering C to IR
  - Passing ill-defined values as function arguments raise UB in C/C++
  - Attaching `noundef` is in progress (mainly by MSan folks)
- (Suggestion) Attach `!noundef` metadata to instructions
  - Certain erroneous operations raise UB in C/C++
  - e.g., Signed overflow, OOB pointer, Loading ill-defined values of non-char type

https://reviews.llvm.org/D81678

2. Improve Undef/Poison Analysis

```
@f(i32 %n) {
loop:
poison = phi [0,    %entry]
             [%i’, %loop]
  %i’
  %cmp   = %i’  <= %n
  br %cmp,
```

poison

poison

poison

poison

UB

Q: Is **%i’** never undef & poison?

A: **Yes**!

(1) non-undef: `%i’` increments from 0

(2) non-poison: “`br %cmp`” raises UB if `poison`.

3. Make More Optimizations Freeze-Aware

- Optimizations
  - SimplifyCFG, InstCombine, InstSimplify
    - Reenable unnecessarily disabled patterns in the presence of freeze.
  - Vectorizer
    - Update vectorization algorithms to handle freeze
- Analyses
  - Freeze makes difference between Must & May Analyses
    - Holds for: one of possible values vs. all possible values

https://reviews.llvm.org/D75808
https://reviews.llvm.org/D87445

Non-Undef/Poison Assumption From Source is Helpful

- Baseline: Fix 16 more bugs by inserting freeze or conditionally enabling it
- Attach `noundef` to function args & `!noundef` to value read when lowering from C/C++
- Run SPEC CPU2017 with –O3, count the unremoved freeze insts.

SPEC CPU2017	Base	Add noundef to fn args	Add noundef to fn args & var reads
# of freeze insts.	42K	36K (86%)	24K (57%)
# of freeze per bench.		49 ~ 95% (Avg. 77%)	27 ~ 80% (Avg. 51%)

How to Write Safe Optimizations

1. Keep in mind that input values can be `undef` or `poison`
2. Be aware that two uses of the same variable may yield different values
   - Ex) `x * 2` ✗ `x + x`
3. Be careful not to introduce new `undef` or `poison` values
   - Ex) `(x +nsw y) +nsw z` ✗ `x +nsw (y +nsw z)`

Making Things Simpler by Removing `undef`

- `undef` is hard to reason about due to partially undefined values
- Alive2 detected >30 miscompilations only caused by `undef`
- Might be possible to use `poison` and `freeze` instead of `undef`

https://bugs.llvm.org/show_bug.cgi?id=33165

Summary

1. LLVM has `undef` and `poison` values
2. Miscompilations can be fixed with freeze by removing corner cases
3. Cost of using freeze has been reduced over time
4. Suggest removing `undef` and using `poison` only