Crellvm: Verified Credible Compilation for LLVM

Seoul National University (Korea)

Jeehoon Kang* **Yoonseung Kim*** Youngju Song*
Juneyoung Lee Sanghoon Park Mark Dongyeon Shin
Yonghyun Kim Sungkeun Cho Joonwon Choi (MIT)
Chung-Kil Hur Kwangkeun Yi

* The first three authors are listed alphabetically.

Reliability of Production Compilers

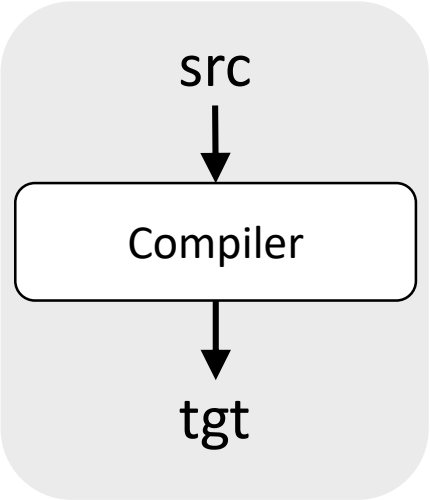
- Stable in most common cases
- Unstable in corner cases
  - Csmith: 79 from GCC / 202 from LLVM
  - EMI: 79 from GCC / 68 from LLVM
- Problematic in practice
  - Low-level systems code

Reliability of Production Compilers

- ➢ Stable in most common cases
- ➢ Unstable in corner cases
  - Csmith: 79 from GCC / 202 from LLVM
  - EMI: 79 from GCC / 68 from LLVM
- ➢ Problematic in practice
  - Low-level systems code

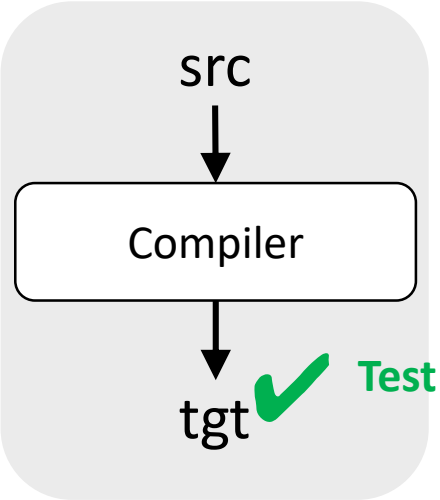
➔ Goal: Improving reliability in corner cases

Approaches to Improving Reliability



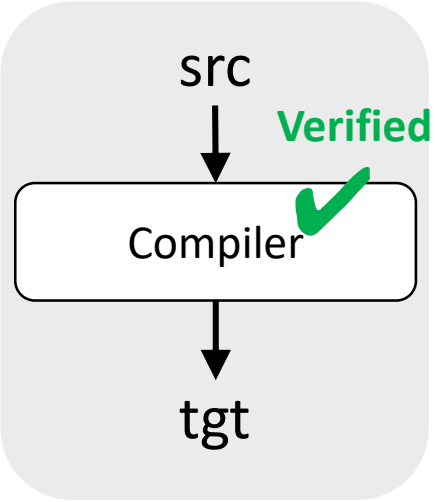
Approaches to Improving Reliability

- (Random) Testing
  - Cannot guarantee high reliability



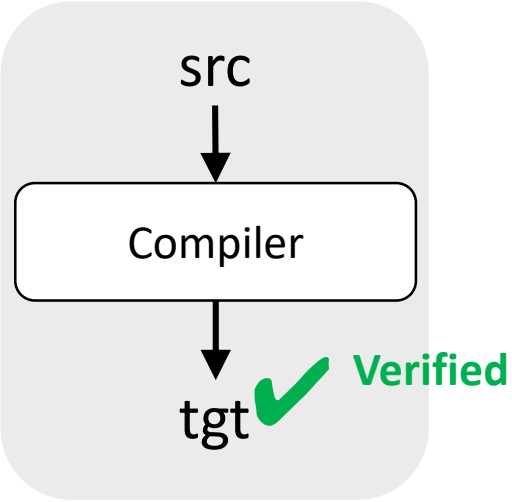
Approaches to Improving Reliability

- (Random) Testing
  - Cannot guarantee high reliability
- Compiler Verification
  - Too expensive to apply to major optimizations of LLVM



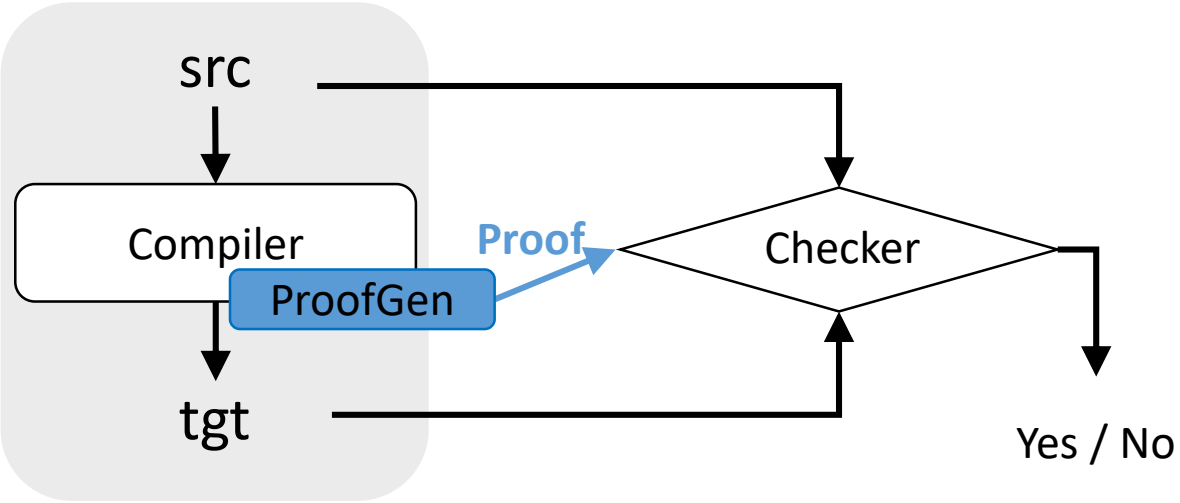
Approaches to Improving Reliability

- (Random) Testing
  - Cannot guarantee high reliability
- Compiler Verification
  - Too expensive to apply to major optimizations of LLVM
- Translation Validation
  - High reliability but not too expensive



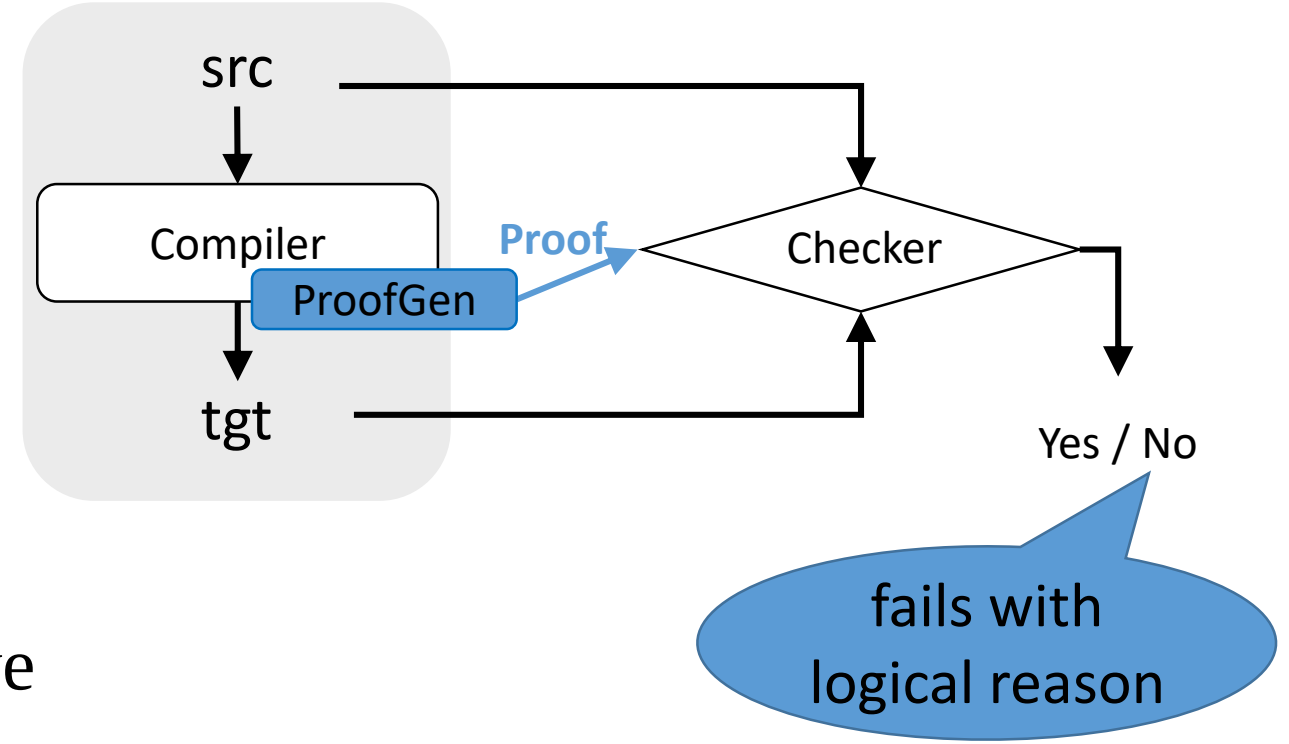
Approaches to Improving Reliability

- (Random) Testing
  - Cannot guarantee high reliability
- Compiler Verification
  - Too expensive to apply to major optimizations of LLVM
- Translation Validation
  - High reliability but not too expensive
  - Credible Compilation
    [Rinard & Marinov 1999]



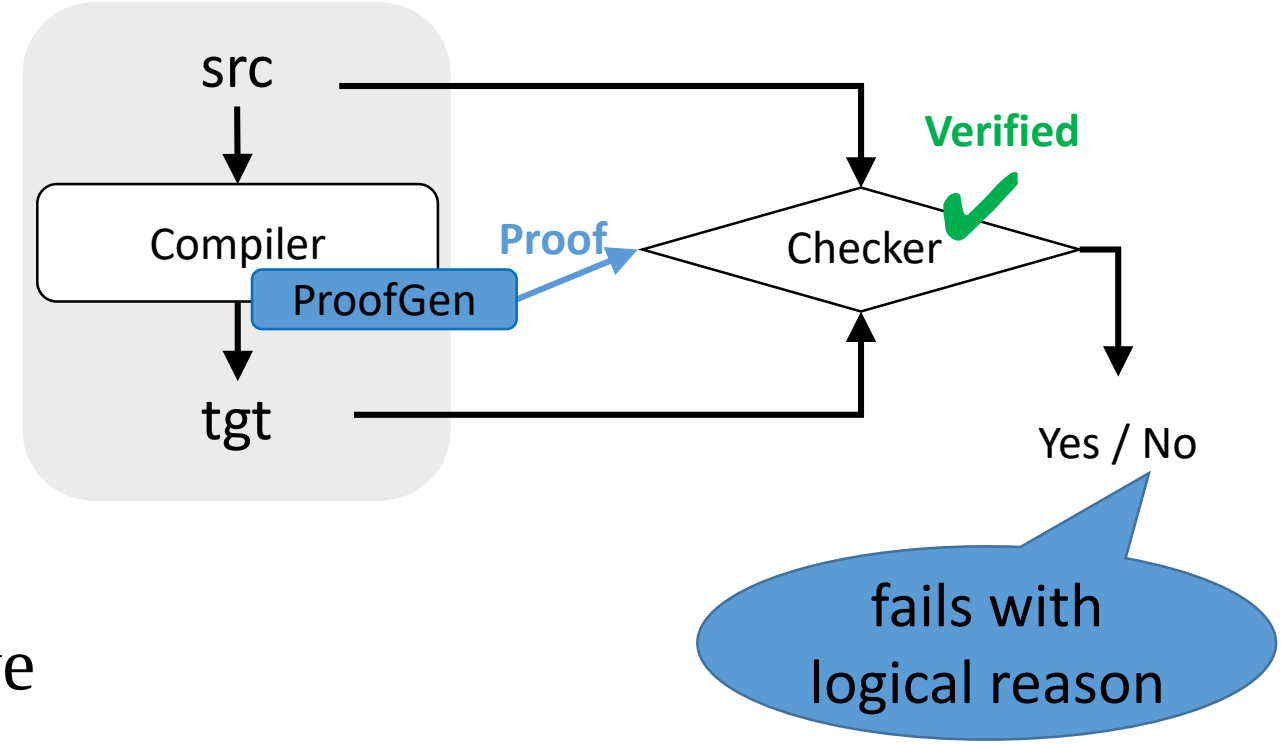
Approaches to Improving Reliability

- (Random) Testing
  - Cannot guarantee high reliability
- Compiler Verification
  - Too expensive to apply to major optimizations of LLVM
- Translation Validation
  - High reliability but not too expensive
  - Credible Compilation
    [Rinard & Marinov 1999]



Approaches to Improving Reliability

- (Random) Testing
  - Cannot guarantee high reliability
- Compiler Verification
  - Too expensive to apply to major optimizations of LLVM
- Translation Validation
  - High reliability but not too expensive
  - Verified Credible Compilation [Rinard & Marinov 1999]



Our Work: Crellvm

- Crellvm
  - Developed a verified credible compilation framework for LLVM
  - Designed a logic specialized for translation validation
  - Verified its proof checker in Coq
- Case studies
  - 3 major optimizations: `mem2reg, gvn, licm`
  - >100 peephole optimizations: `instcombine`
- Result
  - Found 4 long-standing miscompilation bugs (all confirmed, 3 fixed)

Example: A Bug We Found in mem2reg

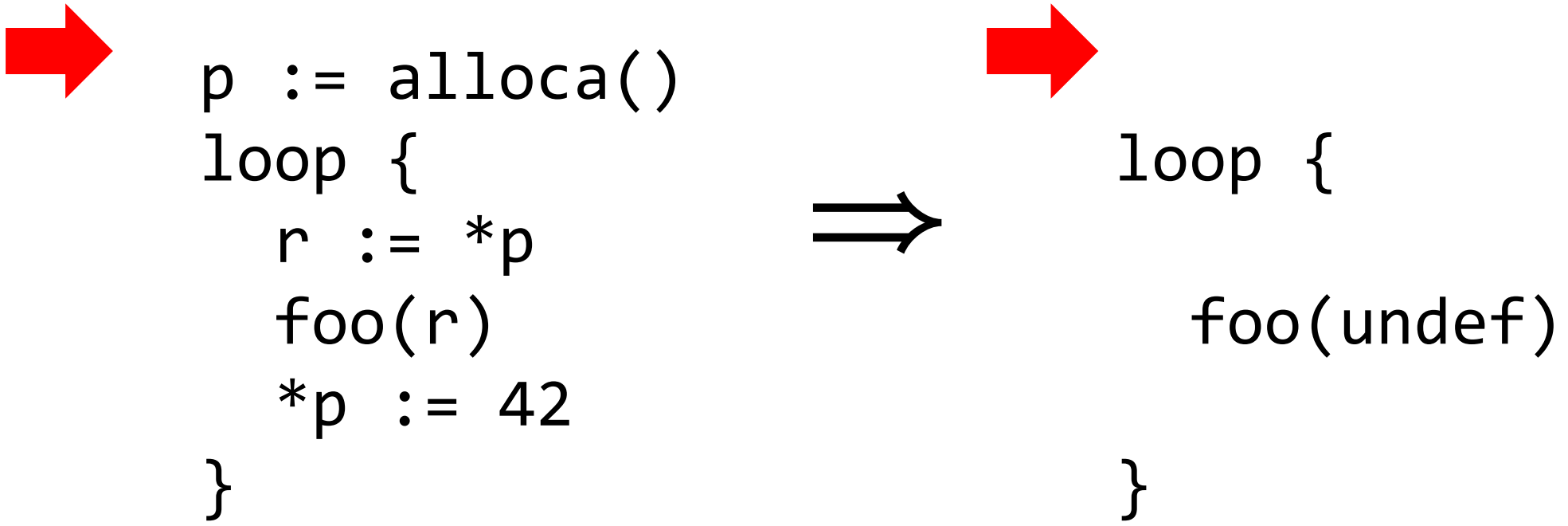
- Credible compilation may detect bugs that testing misses.
- Simplified code from SPEC Benchmark:

```
p := alloca()
loop {
  r := *p
  foo(r)
  *p := 42
}
```

```
loop {

  foo(undef)

}
```

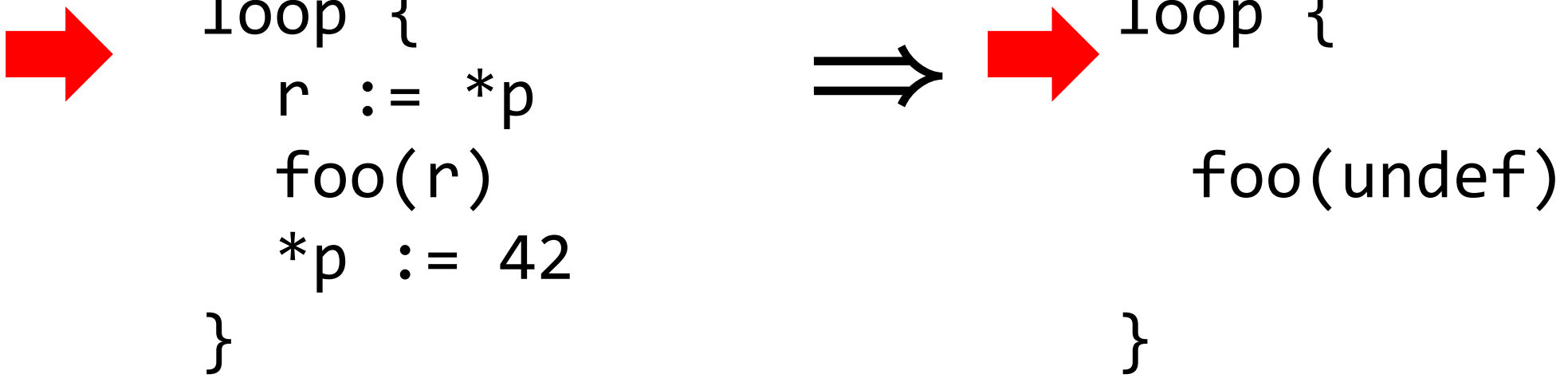
Example: A Bug We Found in mem2reg

- Credible compilation may detect bugs that testing misses.
- Simplified code from SPEC Benchmark:



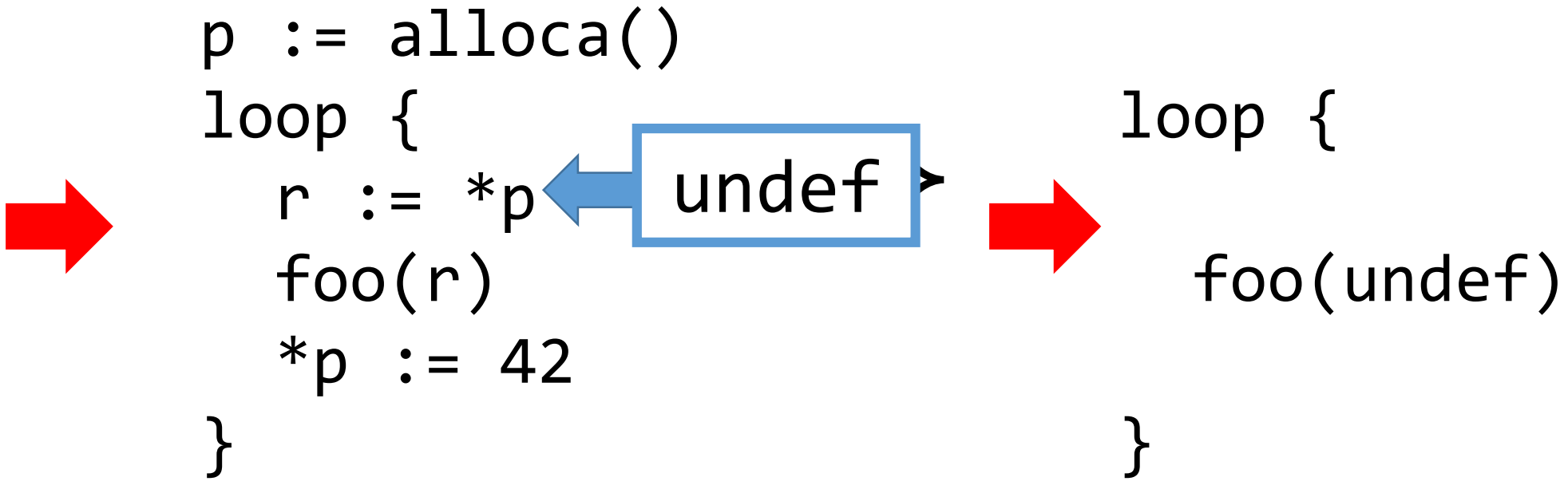
Example: A Bug We Found in mem2reg

- Credible compilation may detect bugs that testing misses.
- Simplified code from SPEC Benchmark:



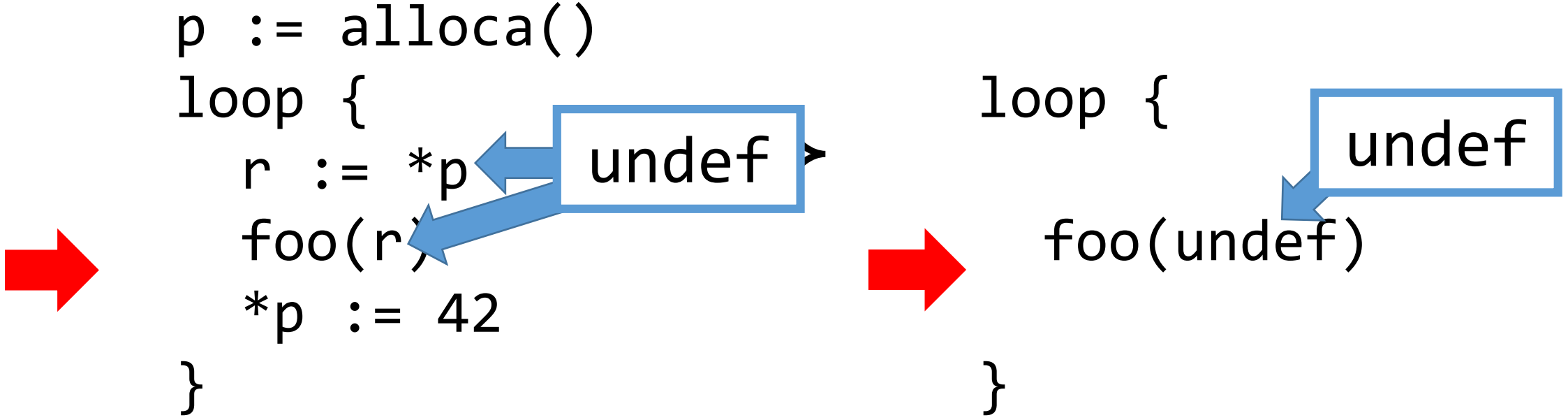
Example: A Bug We Found in mem2reg

- Credible compilation may detect bugs that testing misses.
- Simplified code from SPEC Benchmark:



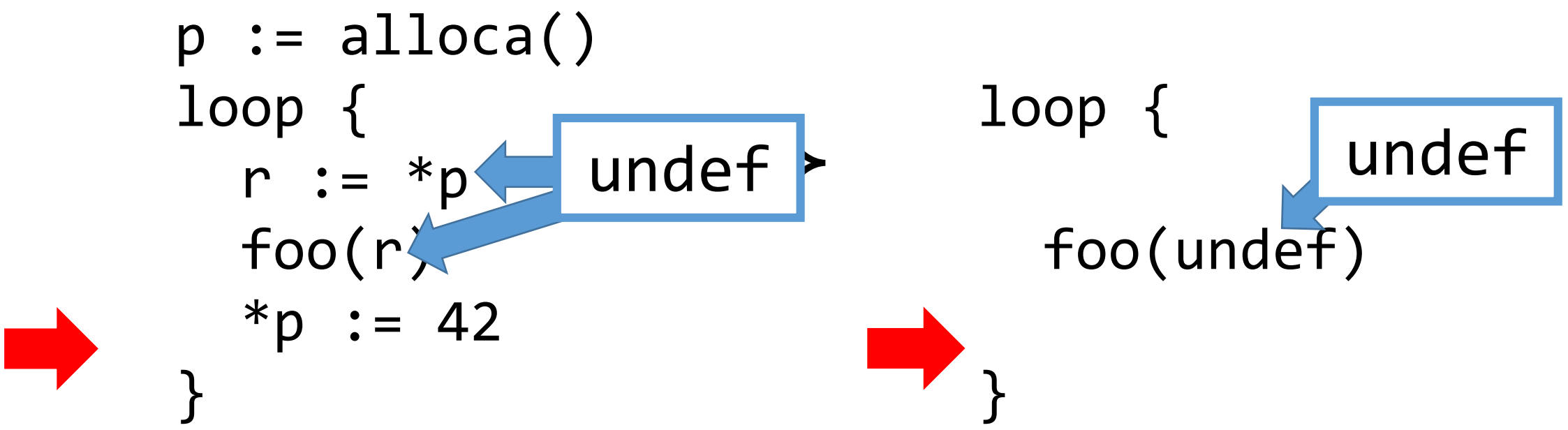
Example: A Bug We Found in mem2reg

- Credible compilation may detect bugs that testing misses.
- Simplified code from SPEC Benchmark:



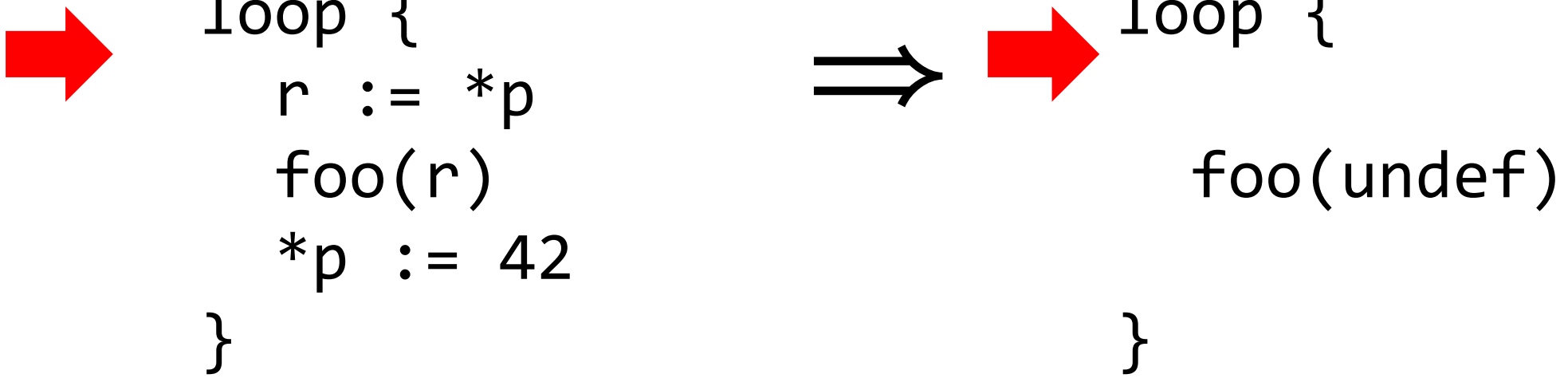
Example: A Bug We Found in mem2reg

- Credible compilation may detect bugs that testing misses.
- Simplified code from SPEC Benchmark:



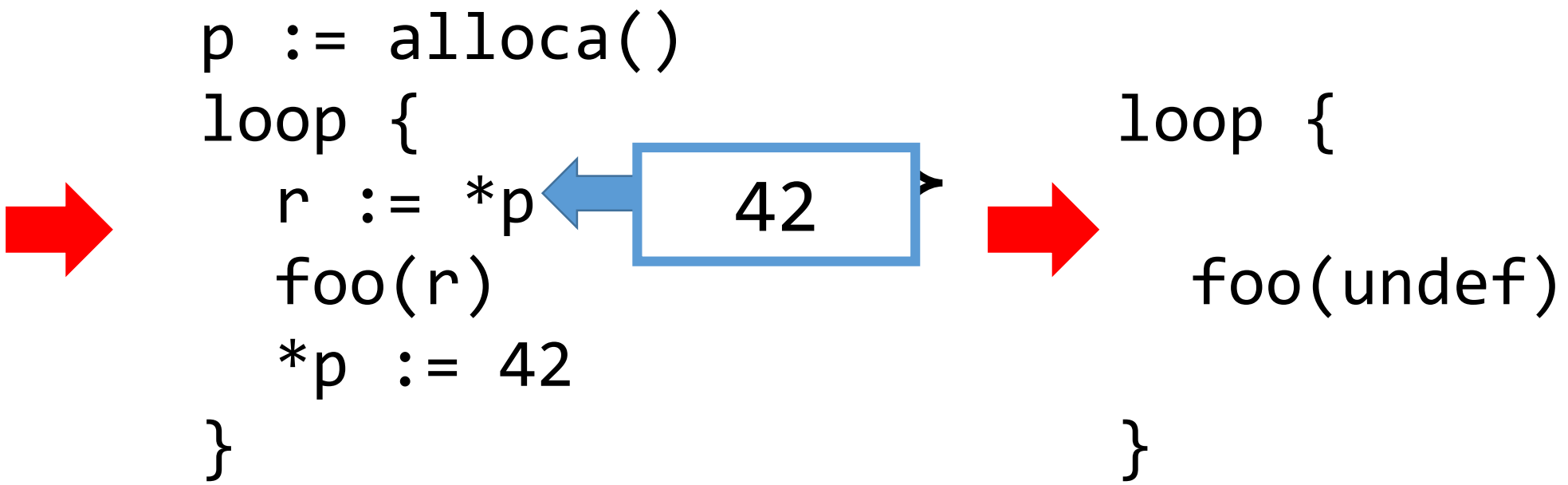
Example: A Bug We Found in mem2reg

- Credible compilation may detect bugs that testing misses.
- Simplified code from SPEC Benchmark:



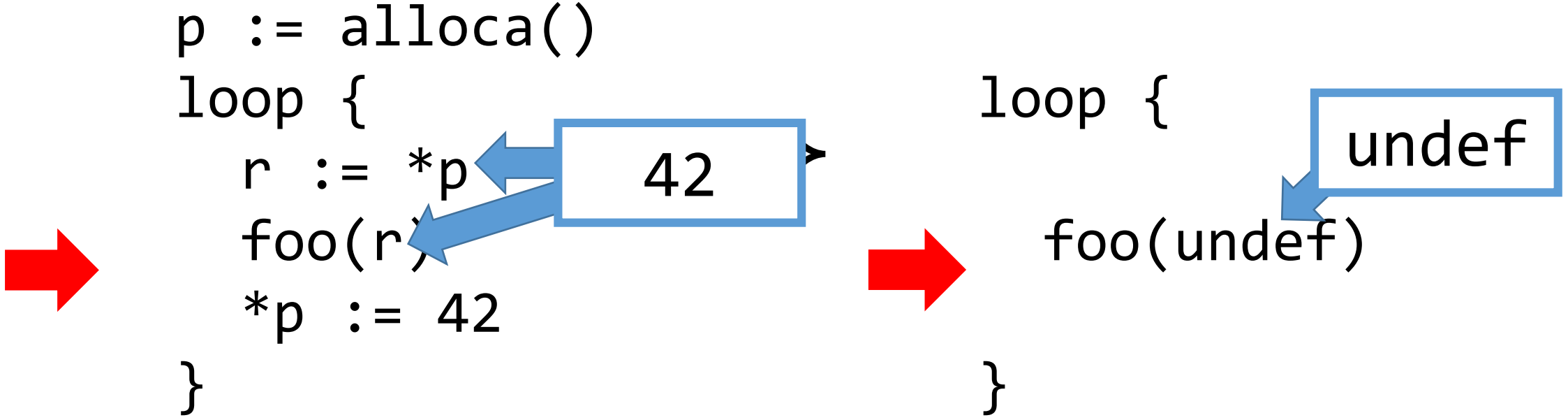
Example: A Bug We Found in mem2reg

- Credible compilation may detect bugs that testing misses.
- Simplified code from SPEC Benchmark:



Example: A Bug We Found in mem2reg

- Credible compilation may detect bugs that testing misses.
- Simplified code from SPEC Benchmark:



Example: A Bug We Found in mem2reg

- Credible
- Simplif

Why testing missed this bug?

Because foo ignores r:

```
foo(r):
  ...
  s = r & 0x0
  ...
```

```
p := all
loop {
  r :=   p
  foo(r)
  *p := 42
}
```

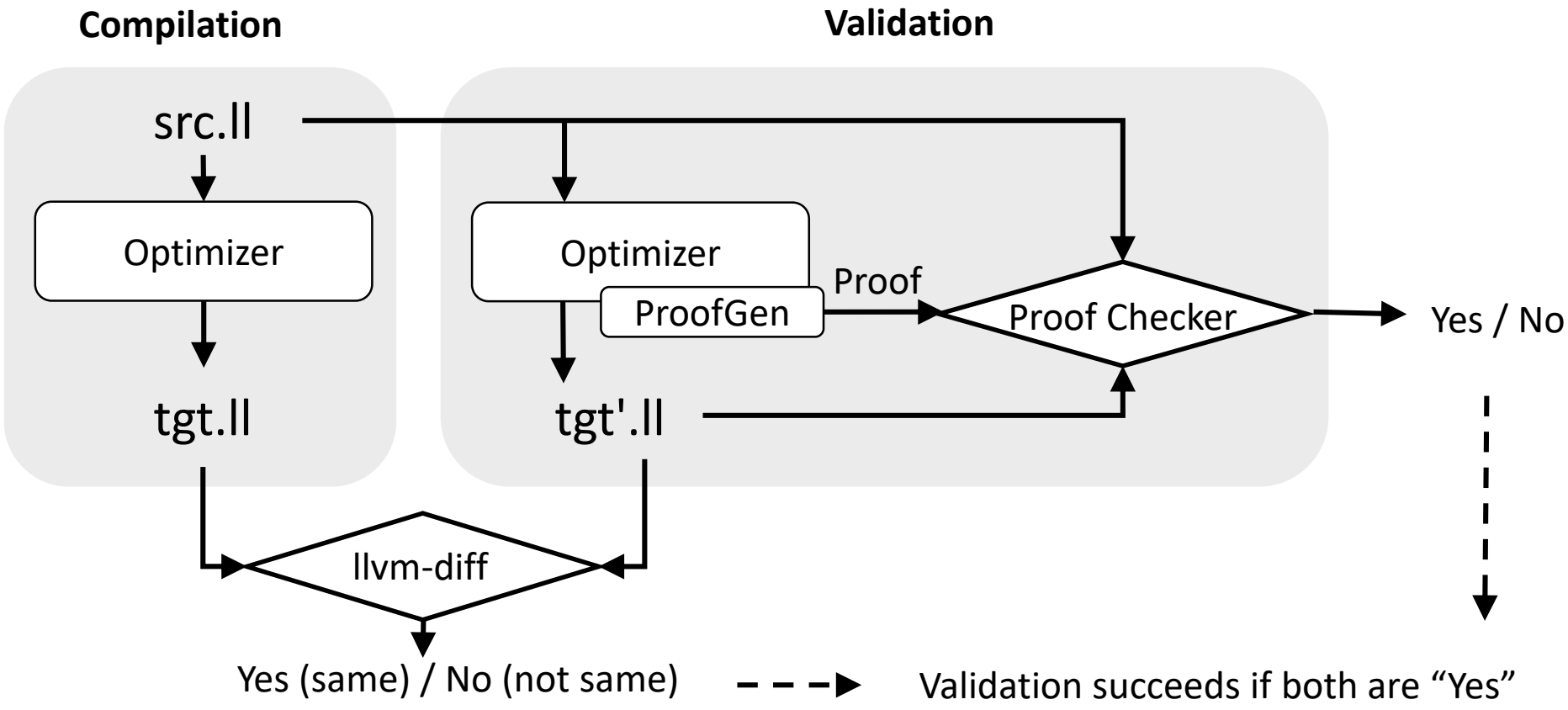
42

```
loop {

  foo(undef)

}
```

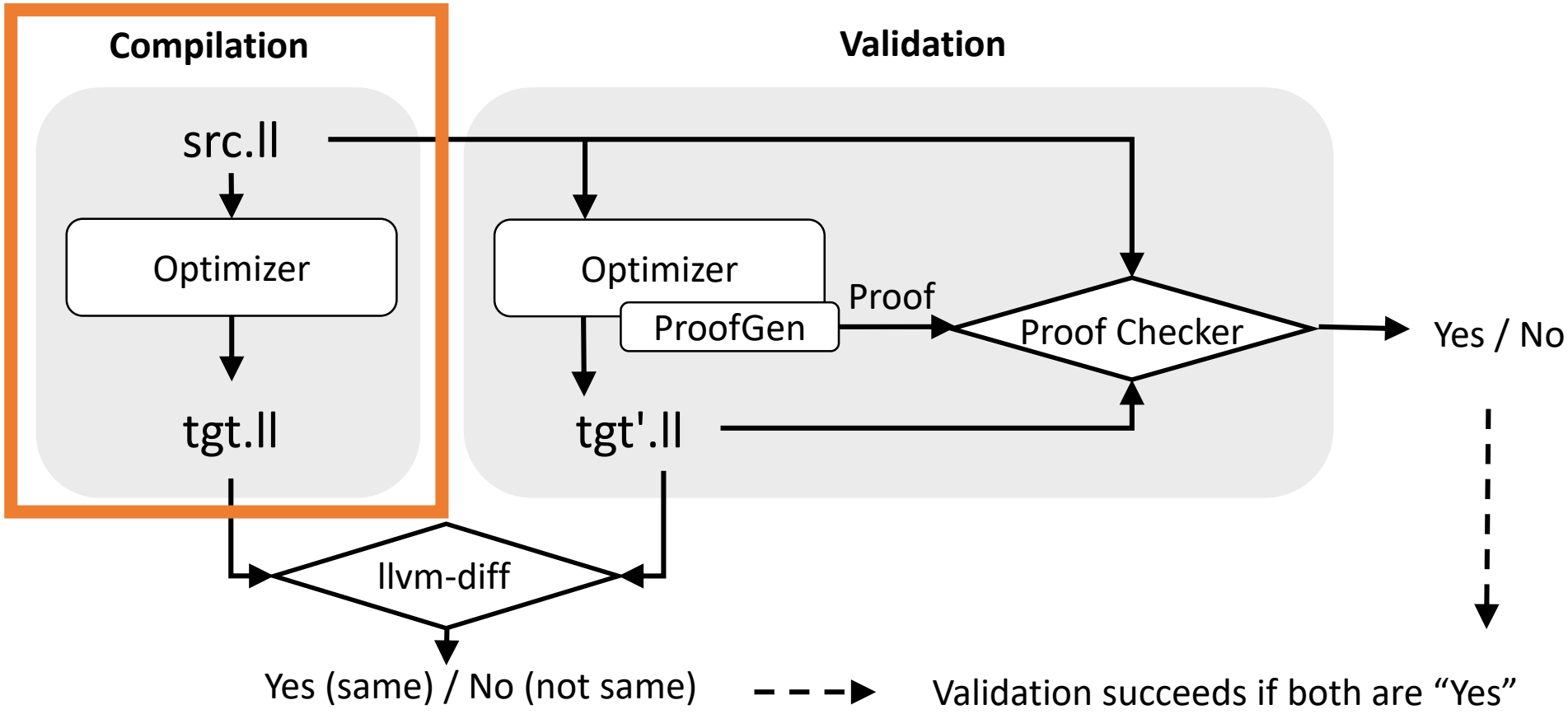
undef

Crellvm framework

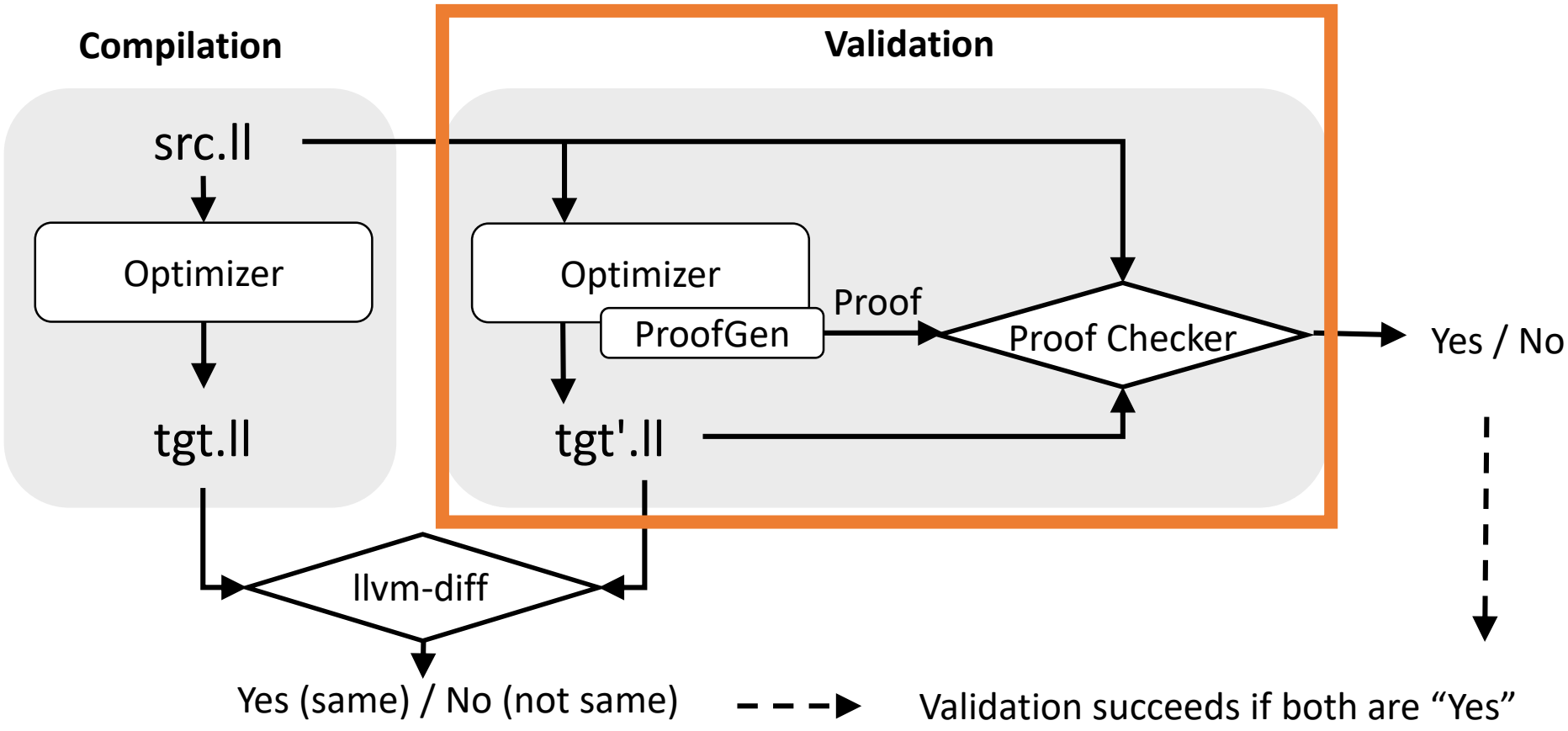
Crellvm Framework



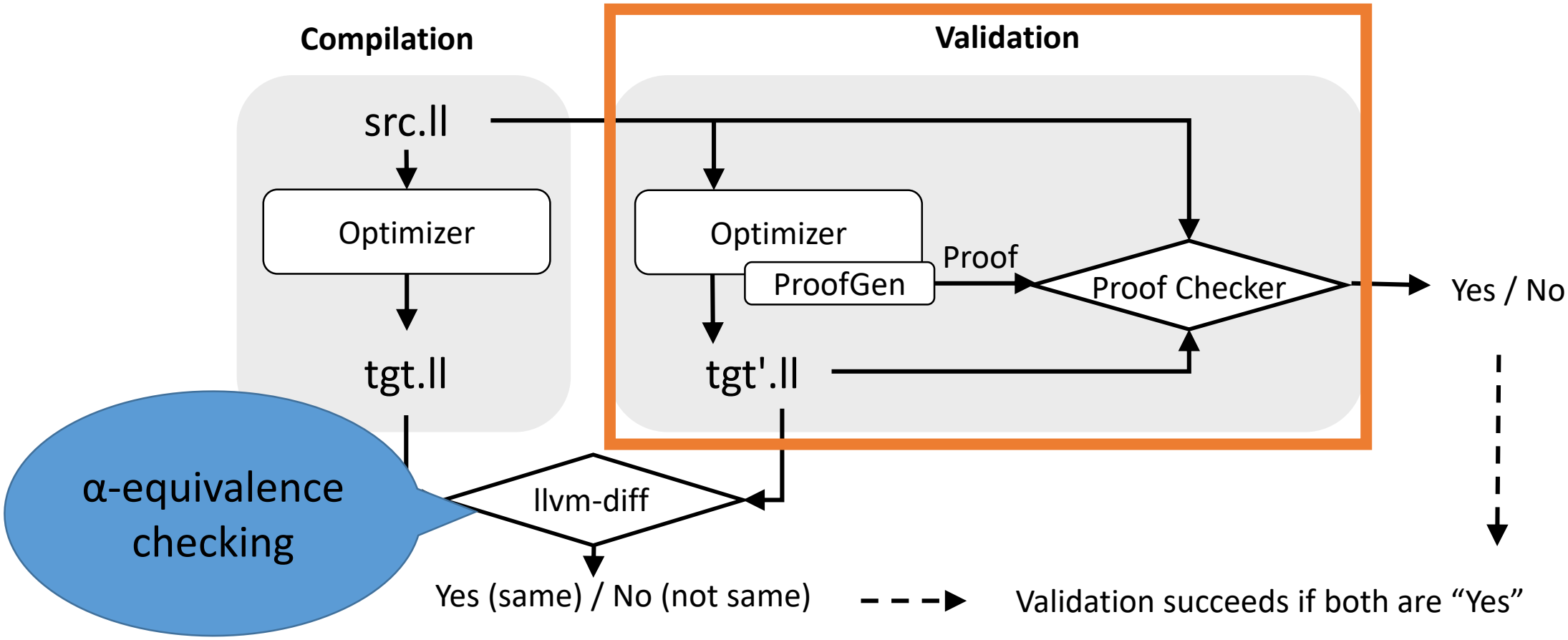
Crellvm Framework



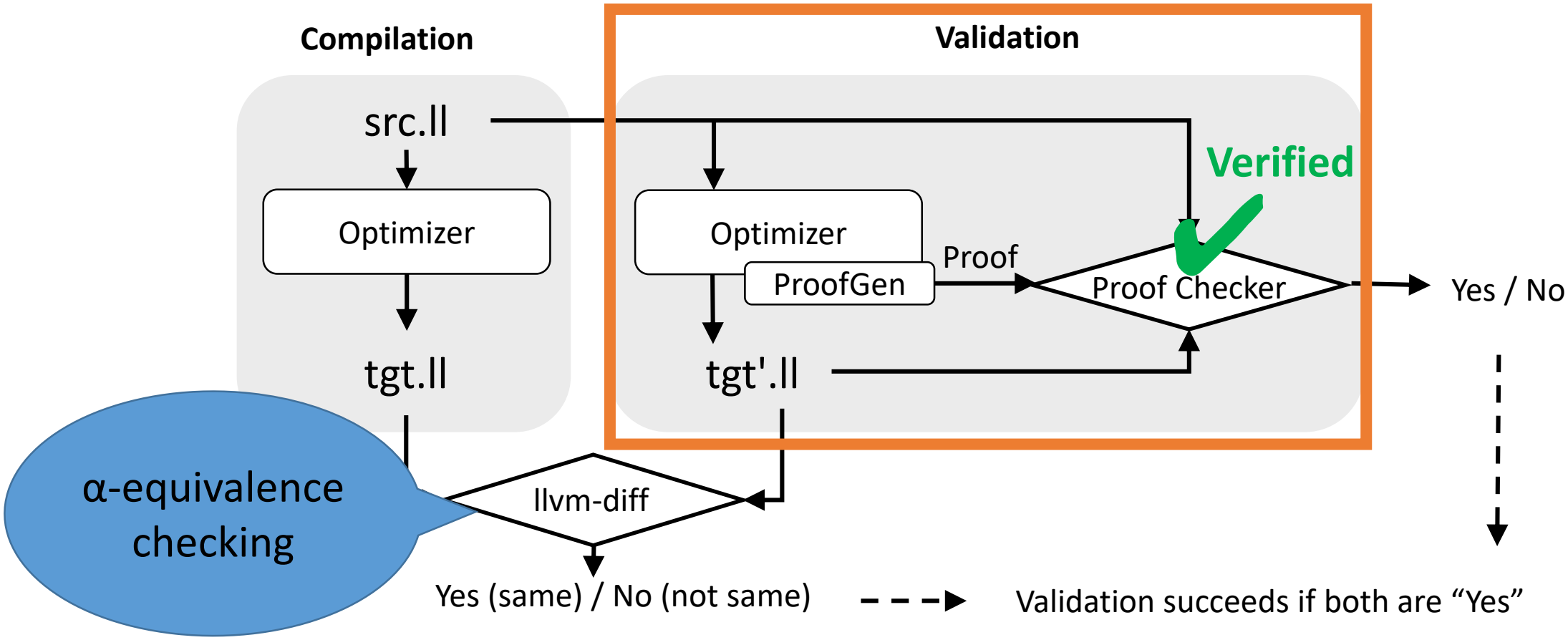
Crellvm Framework



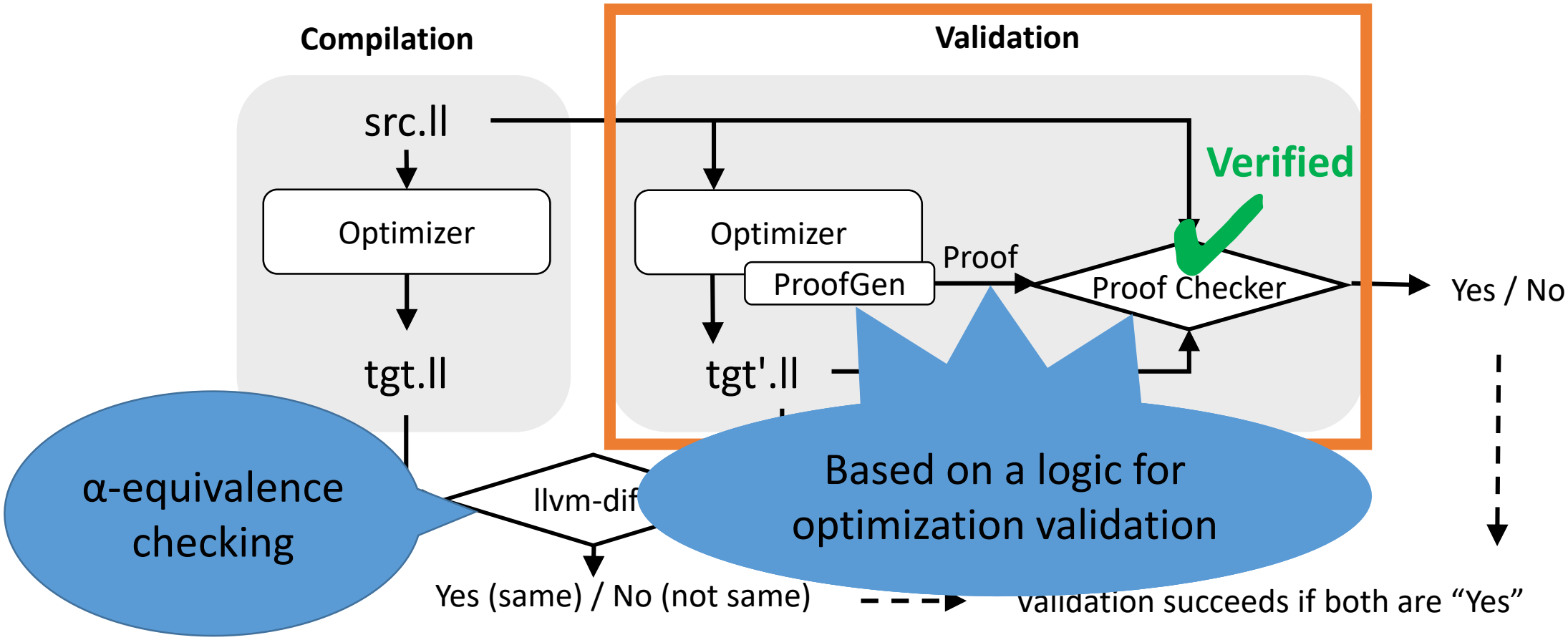
Crellvm Framework



Crellvm Framework

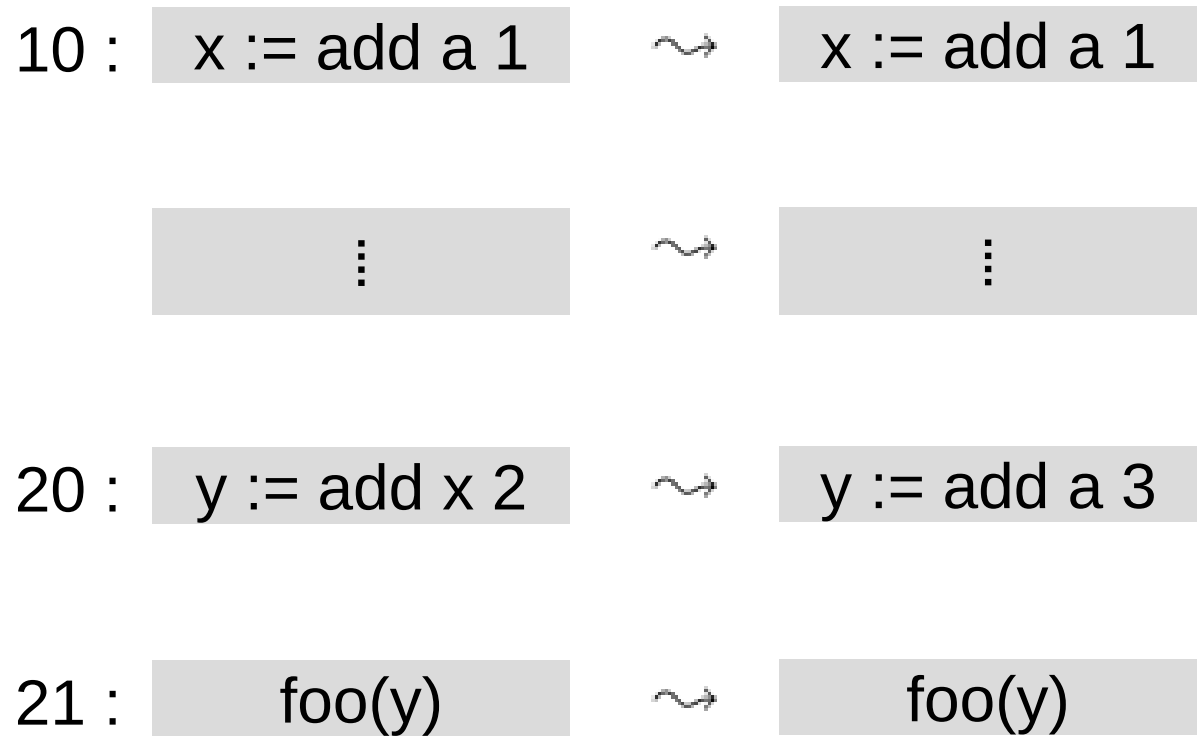


Crellvm Framework



ERHL: A Logic for Optimization Validation

- Assoc-add optimization in `instcombine`



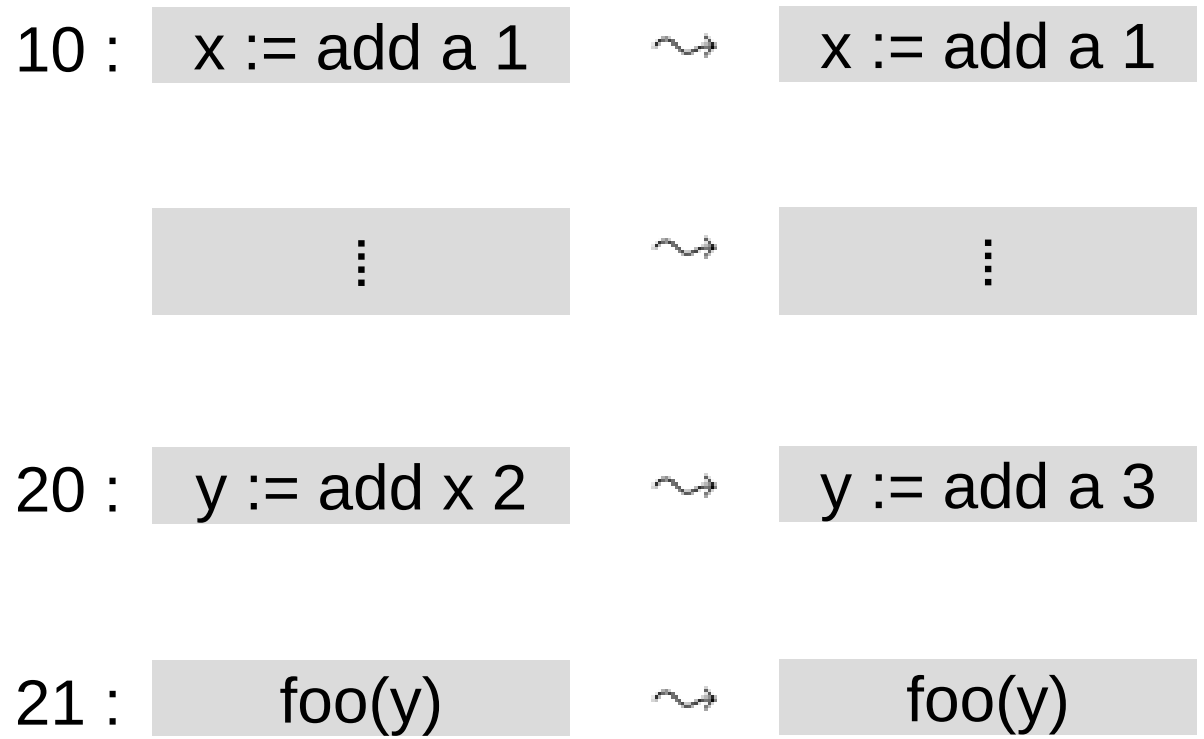
ERHL: A[Extensible Relational Hoare Logic]r Optimization Validation

- Assoc-add in `instcombine`

10 :	x := add a 1	⇝	x := add a 1
	⋮	⇝	⋮
20 :	y := add x 2	⇝	y := add a 3
21 :	foo(y)	⇝	foo(y)

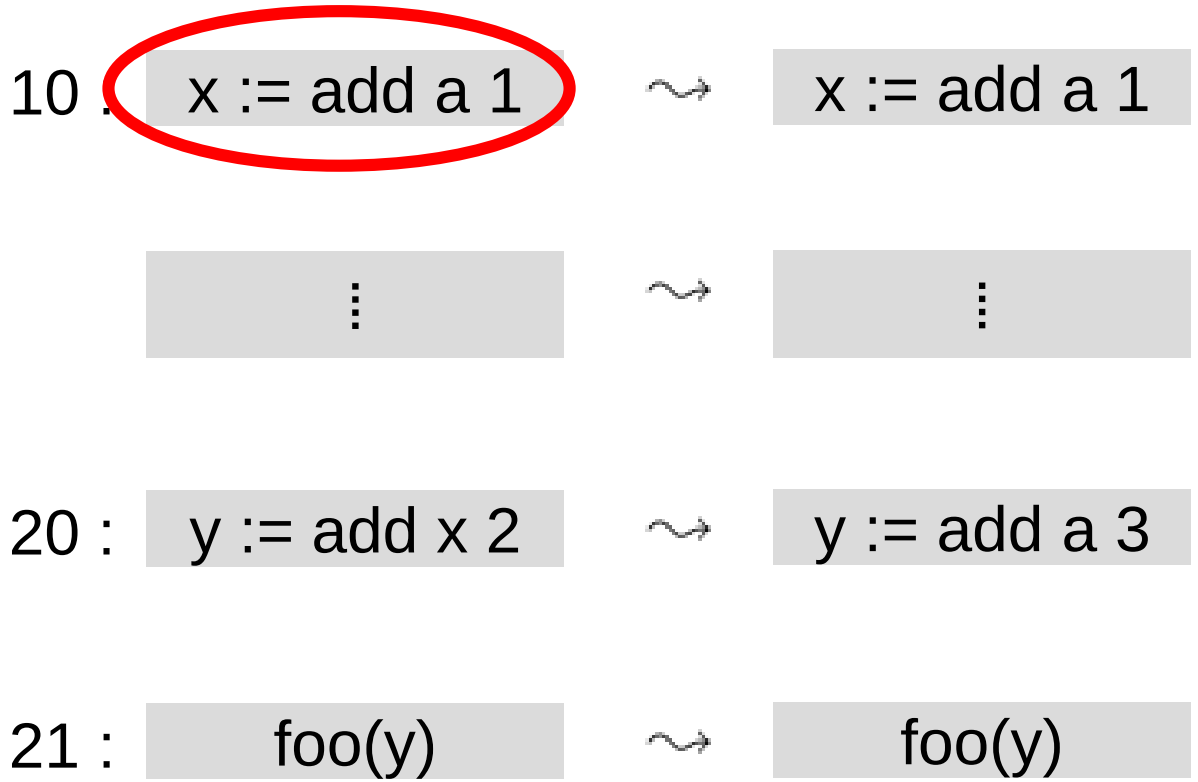
ERHL: A Logic for Optimization Validation

- Assoc-add optimization in `instcombine`



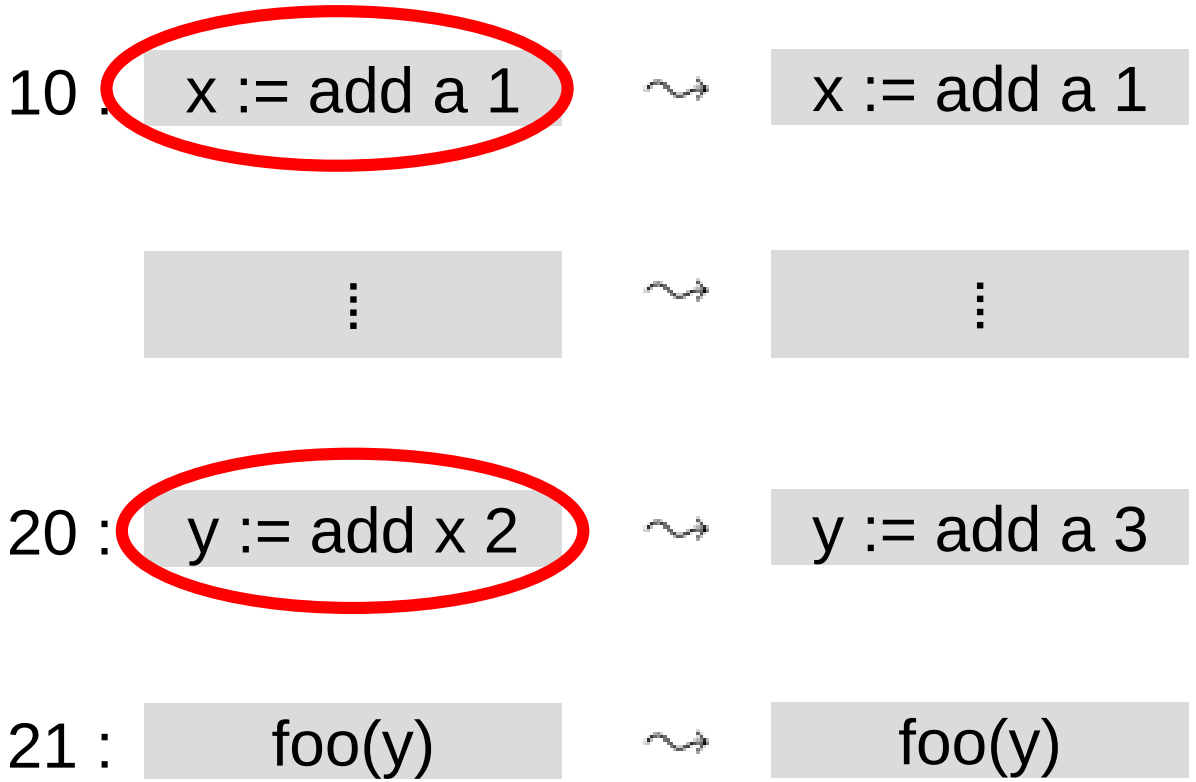
ERHL: A Logic for Optimization Validation

- Assoc-add optimization in `instcombine`



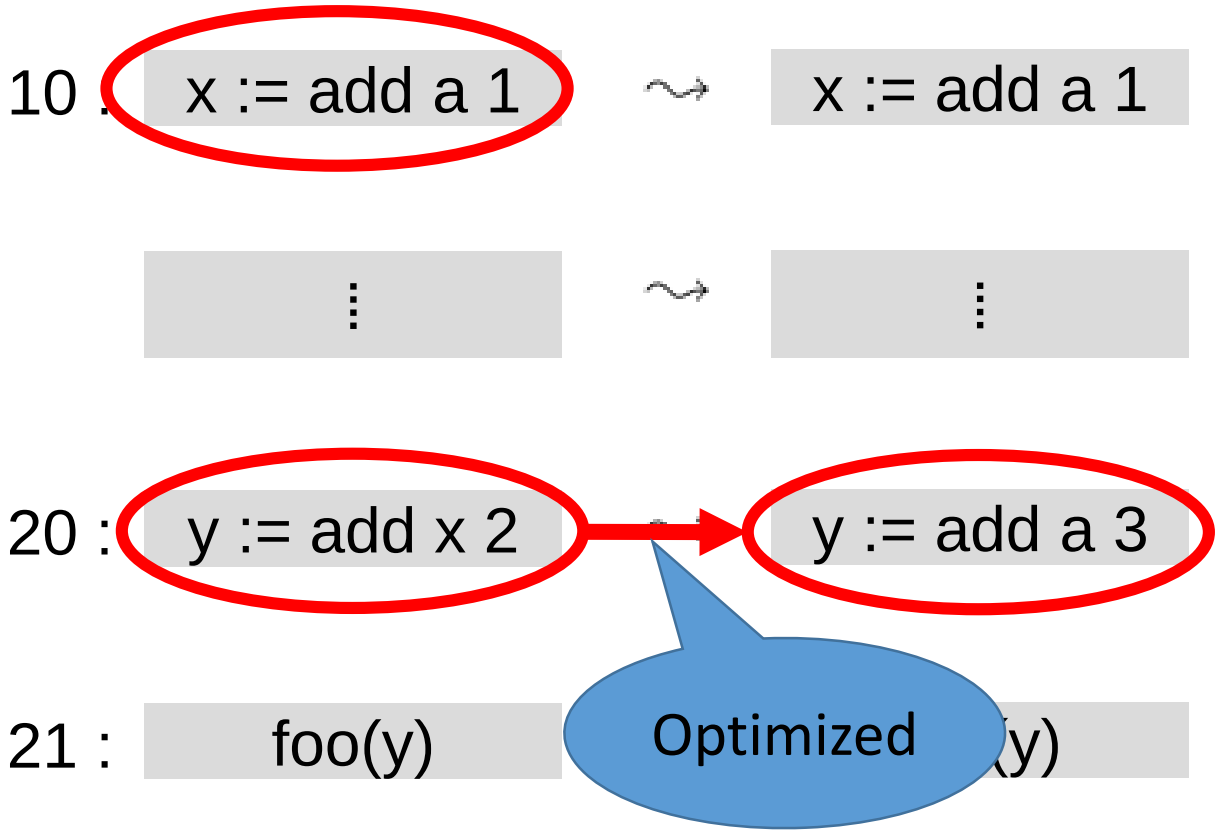
ERHL: A Logic for Optimization Validation

- Assoc-add optimization in `instcombine`



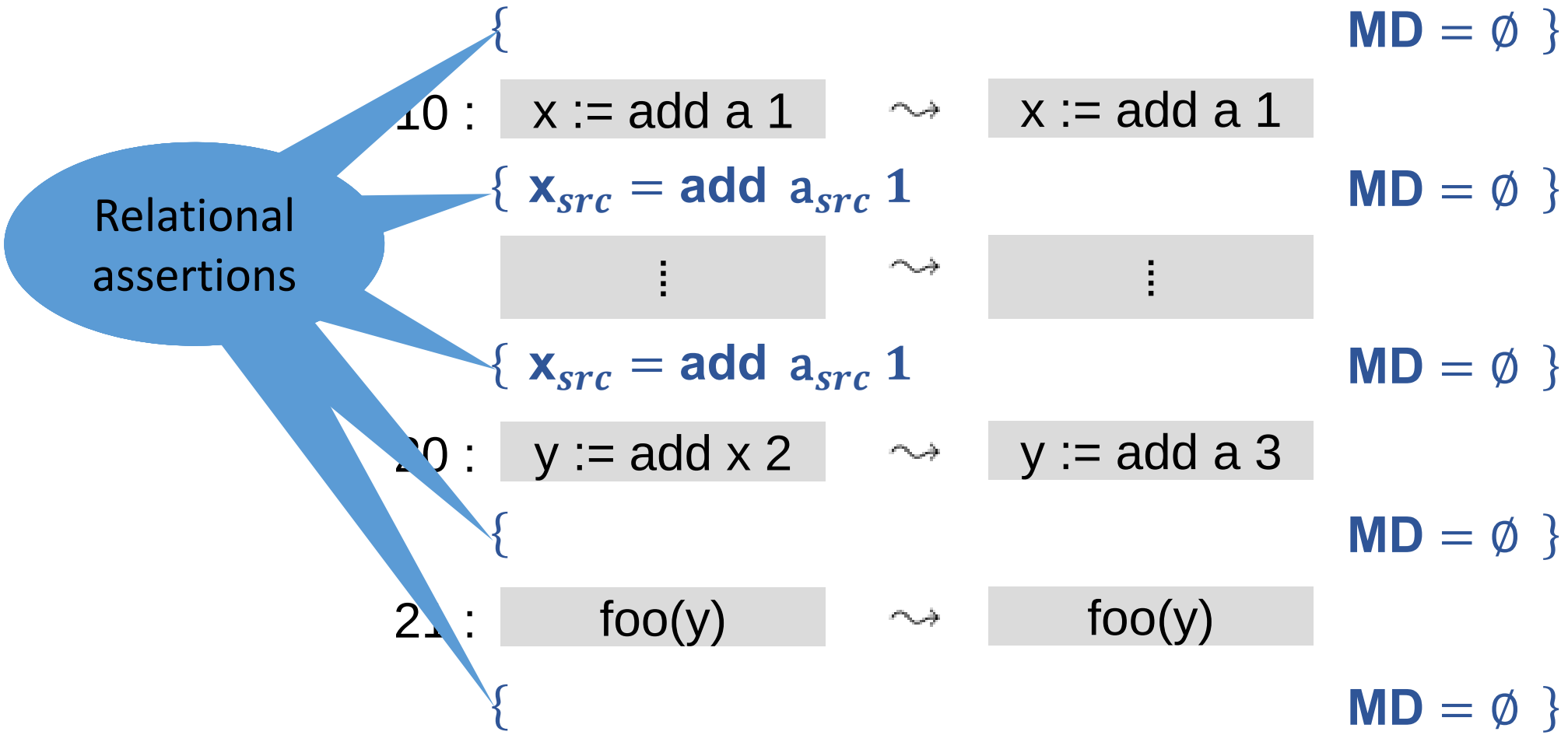
ERHL: A Logic for Optimization Validation

- Assoc-add optimization in `instcombine`



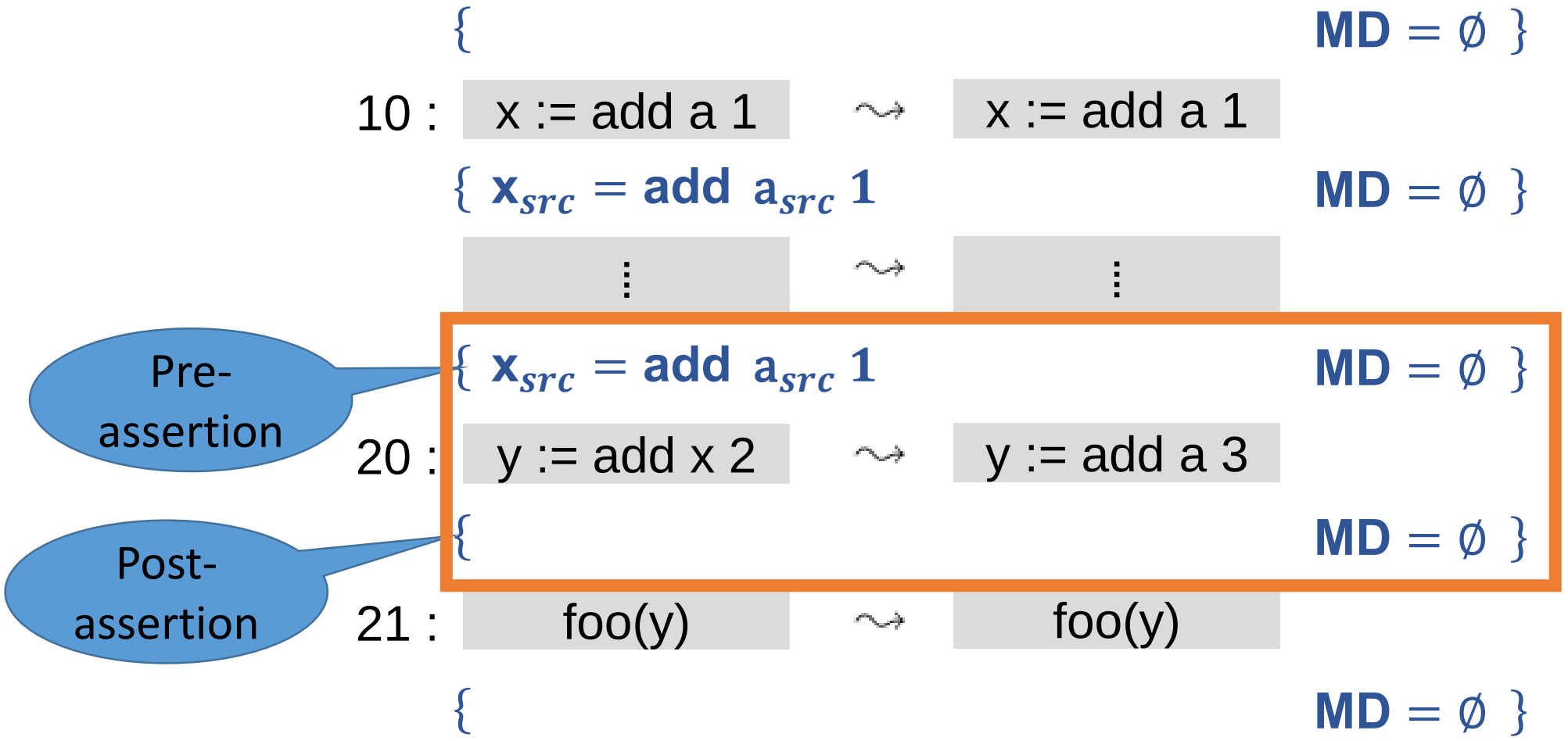
ERHL: A Logic for Optimization Validation

- Assoc-add optimization in `instcombine`



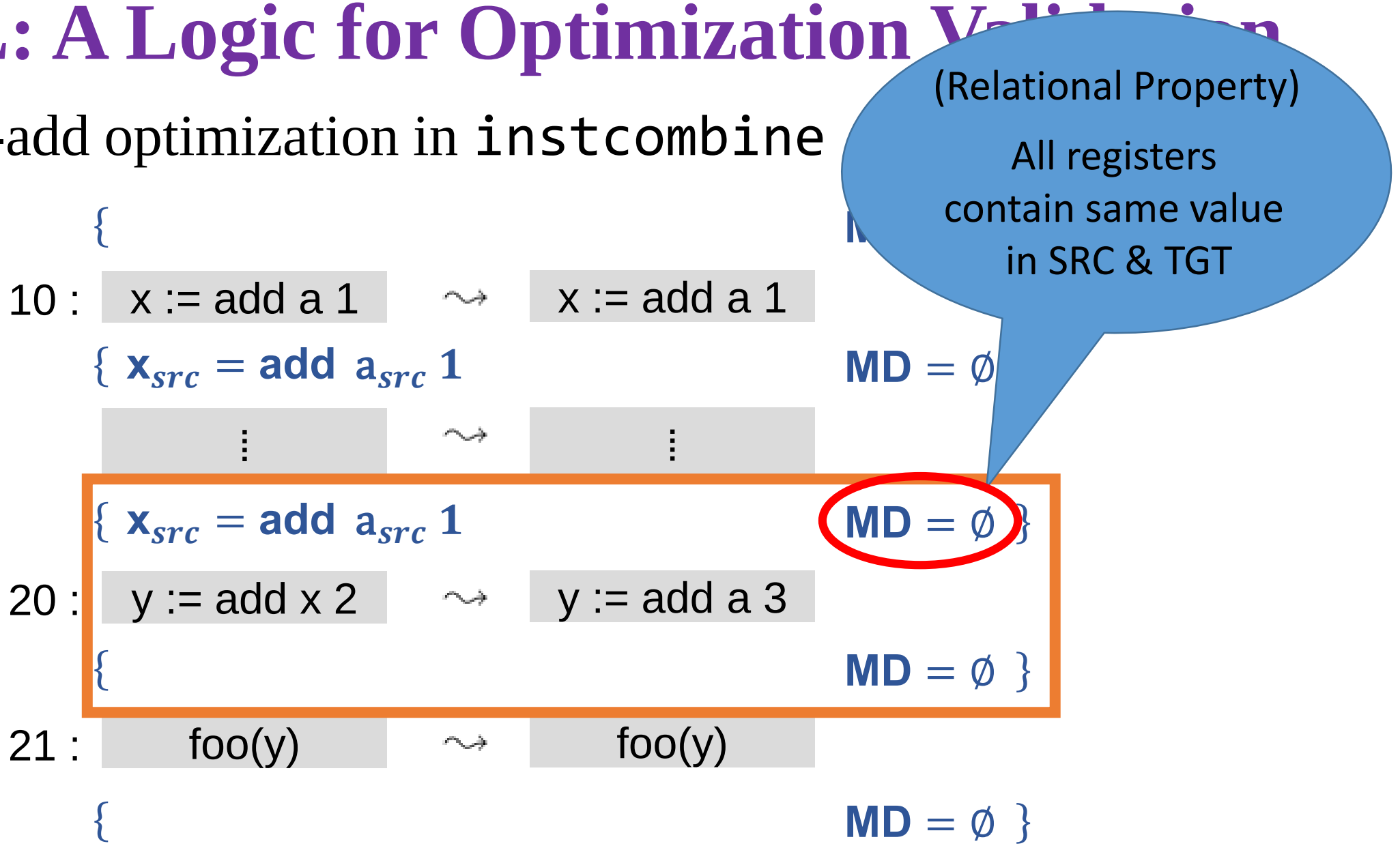
ERHL: A Logic for Optimization Validation

- Assoc-add optimization in `instcombine`



ERHL: A Logic for Optimization Validation

➢ Assoc-add optimization in `instcombine`



ERHL : A Logic for Optimization Validation

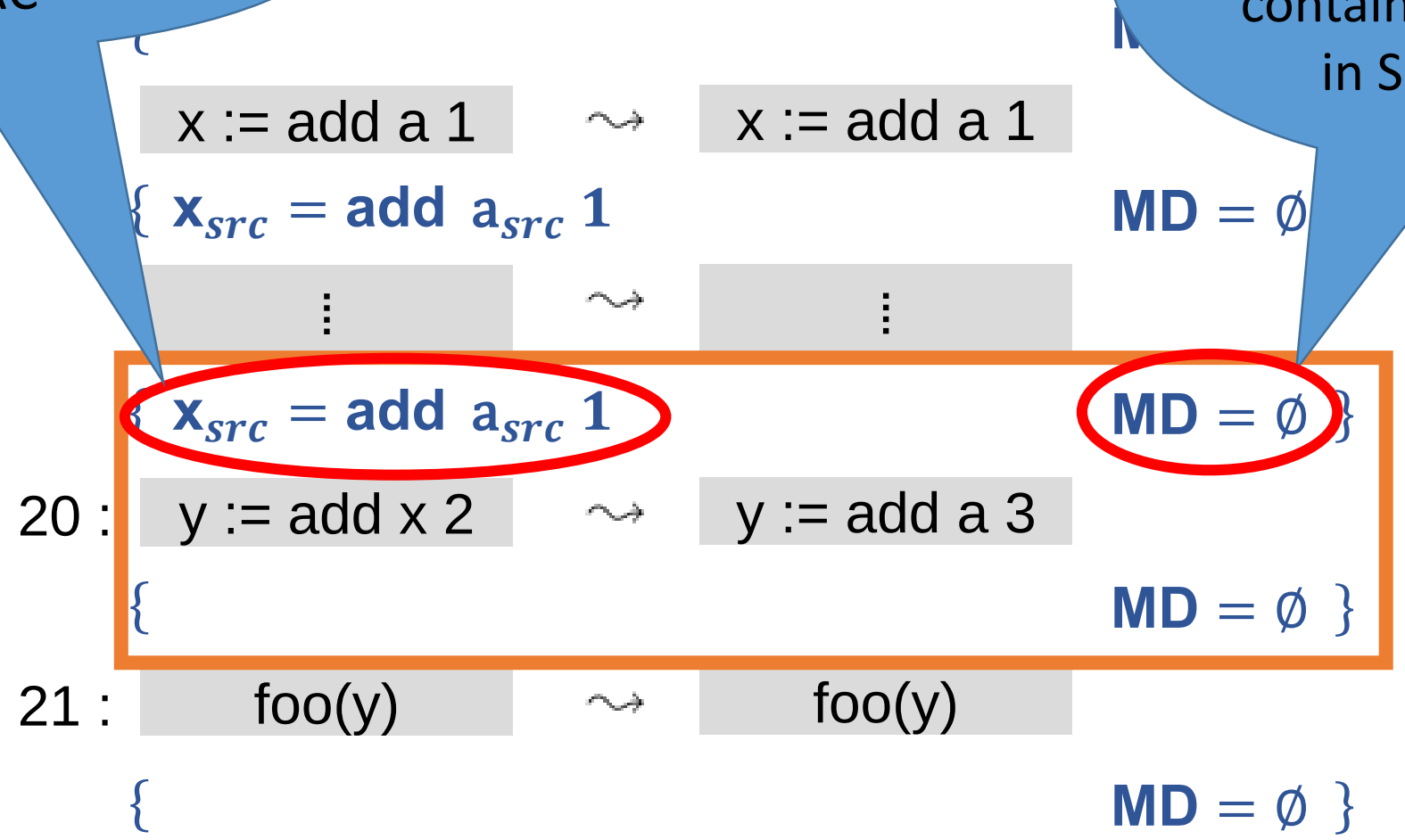
...zation in `instcombine`

(Unary Property)

This equation holds in SRC

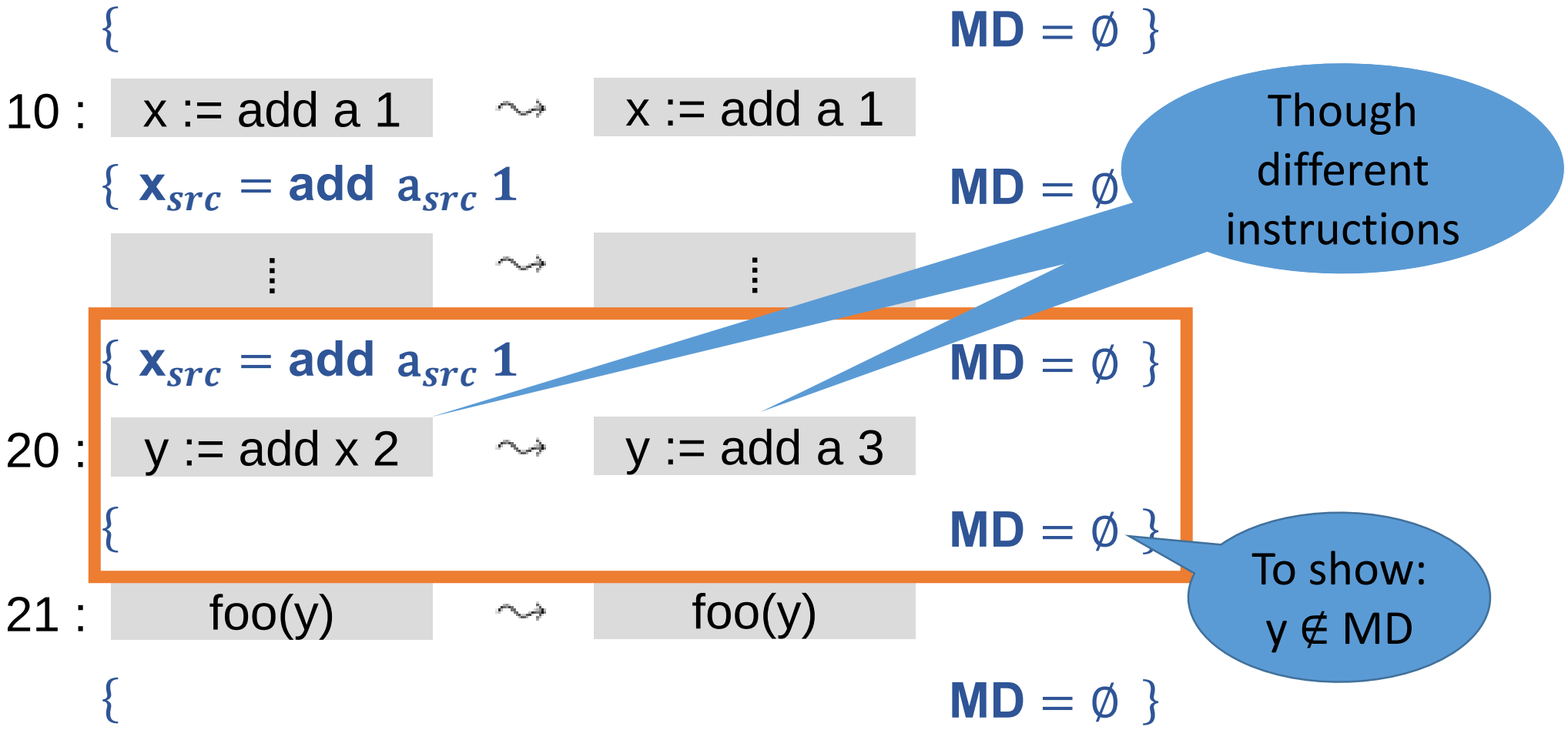
(Relational Property)

All registers contain same value in SRC & TGT



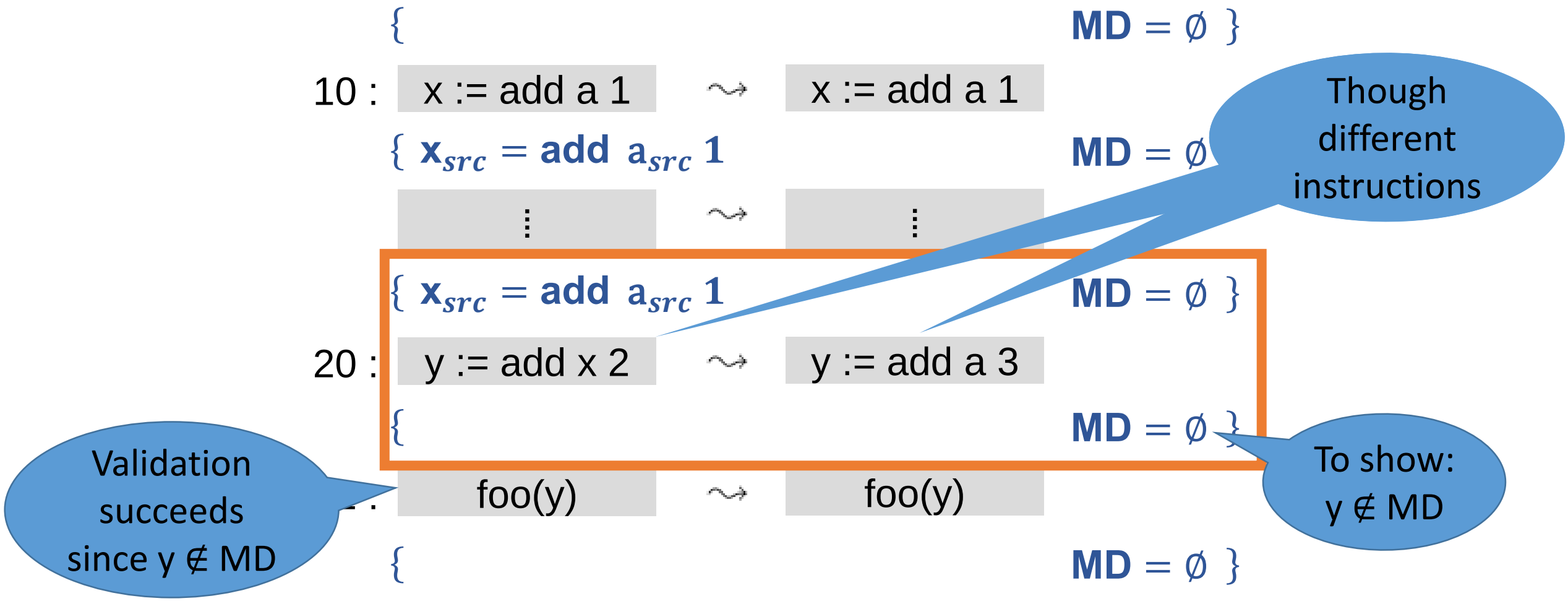
ERHL: A Logic for Optimization Validation

- Assoc-add optimization in `instcombine`



ERHL: A Logic for Optimization Validation

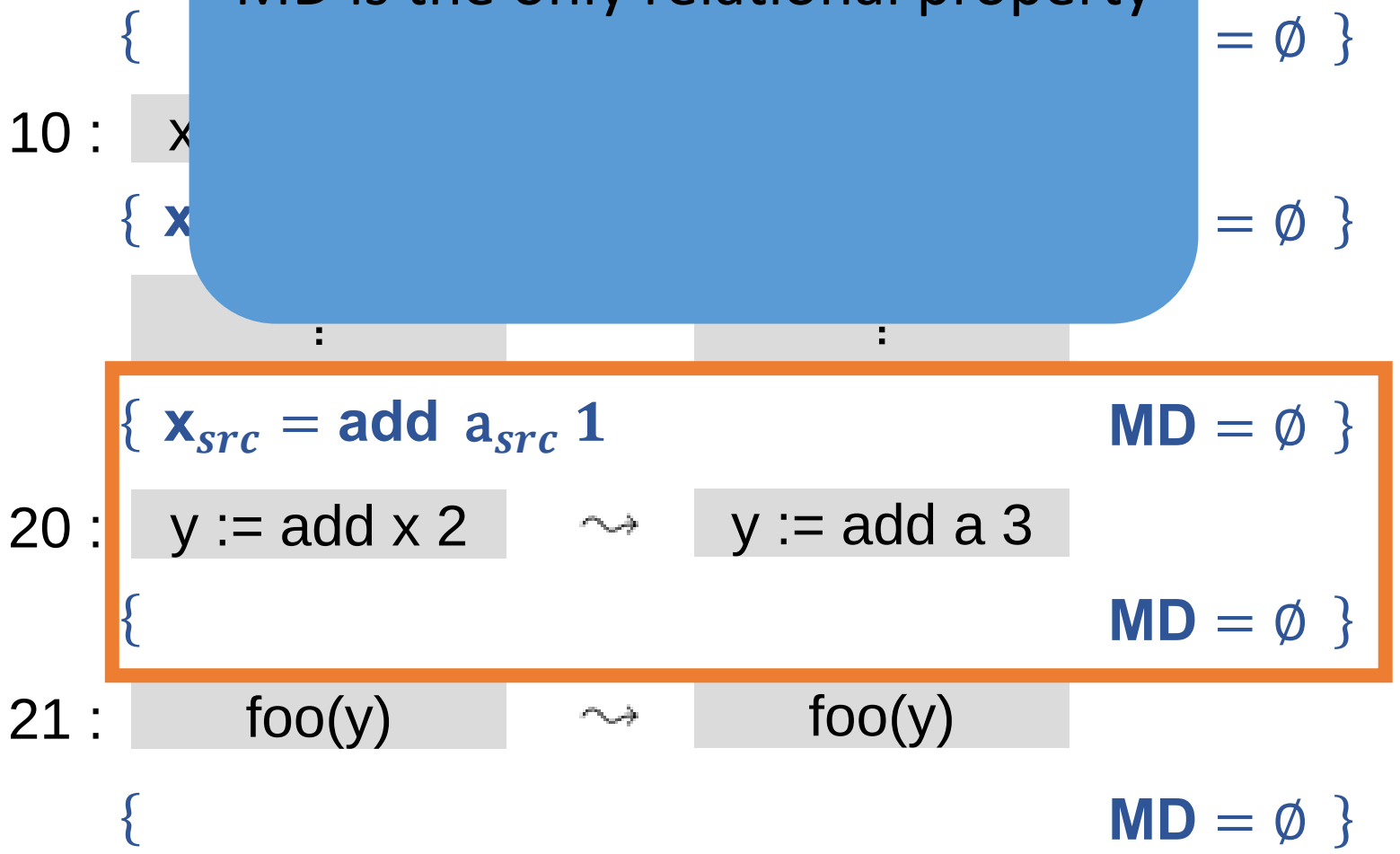
- Assoc-add optimization in `instcombine`



ERHL: A Logic for Optimization Validation

➢ Assoc-add op

(Design Choice)
MD is the only relational property



ERHL: A Logic for Optimization Validation

➢ Assoc-add op

(Design Choice)
MD is the only relational property

- Proof checking is simple

$\{ \quad = \emptyset \}$

10 : x

$\{ \mathbf{x} \quad = \emptyset \}$

:

$\{ \mathbf{x}_{src} = \mathbf{add}\ \mathbf{a}_{src}\ \mathbf{1} \quad \mathbf{MD} = \emptyset \}$

20 : y := add x 2 ⇝ y := add a 3

$\{ \quad \mathbf{MD} = \emptyset \}$

21 : foo(y) ⇝ foo(y)

$\{ \quad \mathbf{MD} = \emptyset \}$

ERHL: A Logic for Optimization Validation

➢ Assoc-add op

(Design Choice)
MD is the only relational property

- Proof checking is simple
- Still expressive enough

{ $= \emptyset$ }

10 : x

{ x $= \emptyset$ }

:

{ $\mathbf{x}_{src} = \mathbf{add}\ \mathbf{a}_{src}\ \mathbf{1}$ **MD** $= \emptyset$ }

20 : y := add x 2 ⇝ y := add a 3

{ **MD** $= \emptyset$ }

21 : foo(y) ⇝ foo(y)

{ **MD** $= \emptyset$ }

ERHL: A Logic for Optimization Validation

$\{ \quad \mathbf{x}_{src} = \mathbf{add}\ \mathbf{a}_{src}\ \mathbf{1}$ $\qquad\qquad$ $\mathbf{MD} = \emptyset \quad \}$

20 : y := add x 2 $\quad \rightsquigarrow \quad$ y := add a 3

$\{$ $\qquad\qquad$ $\mathbf{MD} = \emptyset \quad \}$

ERHL: A Logic for Optimization Validation

Pre-assertion

$\{ \; \mathbf{x}_{src} = \mathbf{add} \; \mathbf{a}_{src} \; \mathbf{1}$ $\qquad\qquad$ $\mathbf{MD} = \emptyset \quad \}$

20 : y := add x 2 $\quad \rightsquigarrow \quad$ y := add a 3

Post-assertion

$\{$ $\qquad\qquad$ $\mathbf{MD} = \emptyset \quad \}$

ERHL: A Logic for Optimization Validation

$\{ \quad \mathbf{x_{src} = add\ a_{src}\ 1} \qquad\qquad\qquad\qquad\qquad\qquad \mathbf{MD = \emptyset} \quad \}$

20 : y := add x 2 $\rightsquigarrow$ y := add a 3

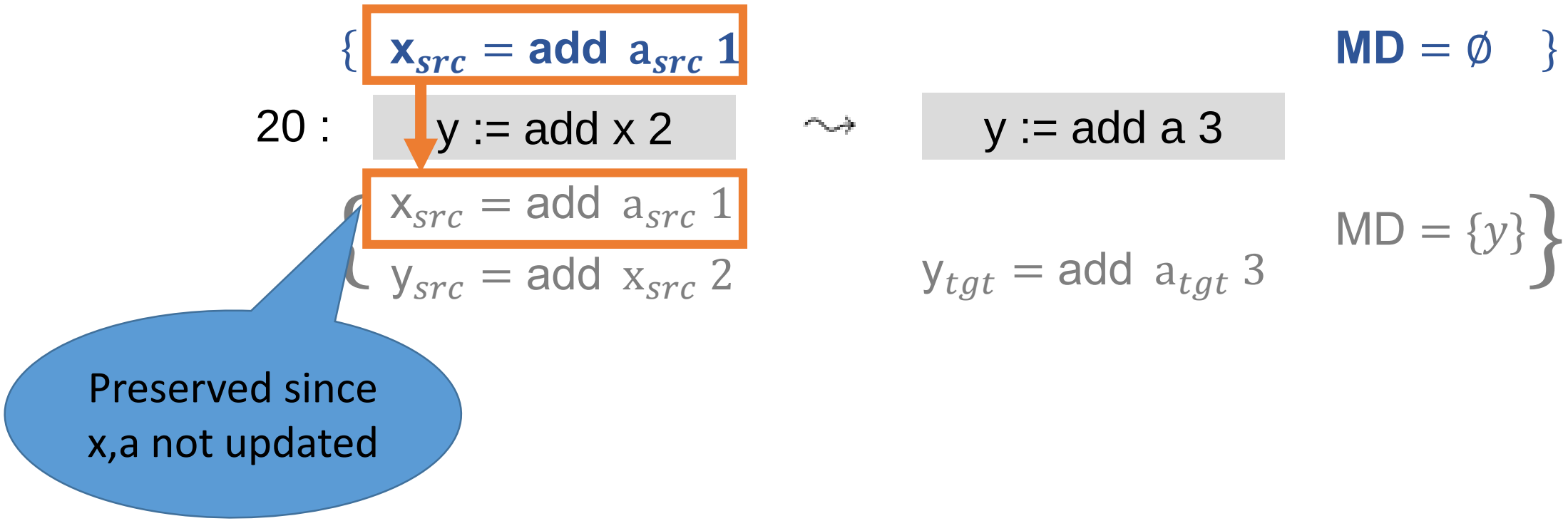
Strong post-condition

$\{ \quad x_{src} = add\ a_{src}\ 1 \qquad\qquad\qquad\qquad\qquad\qquad MD = \{y\} \}$

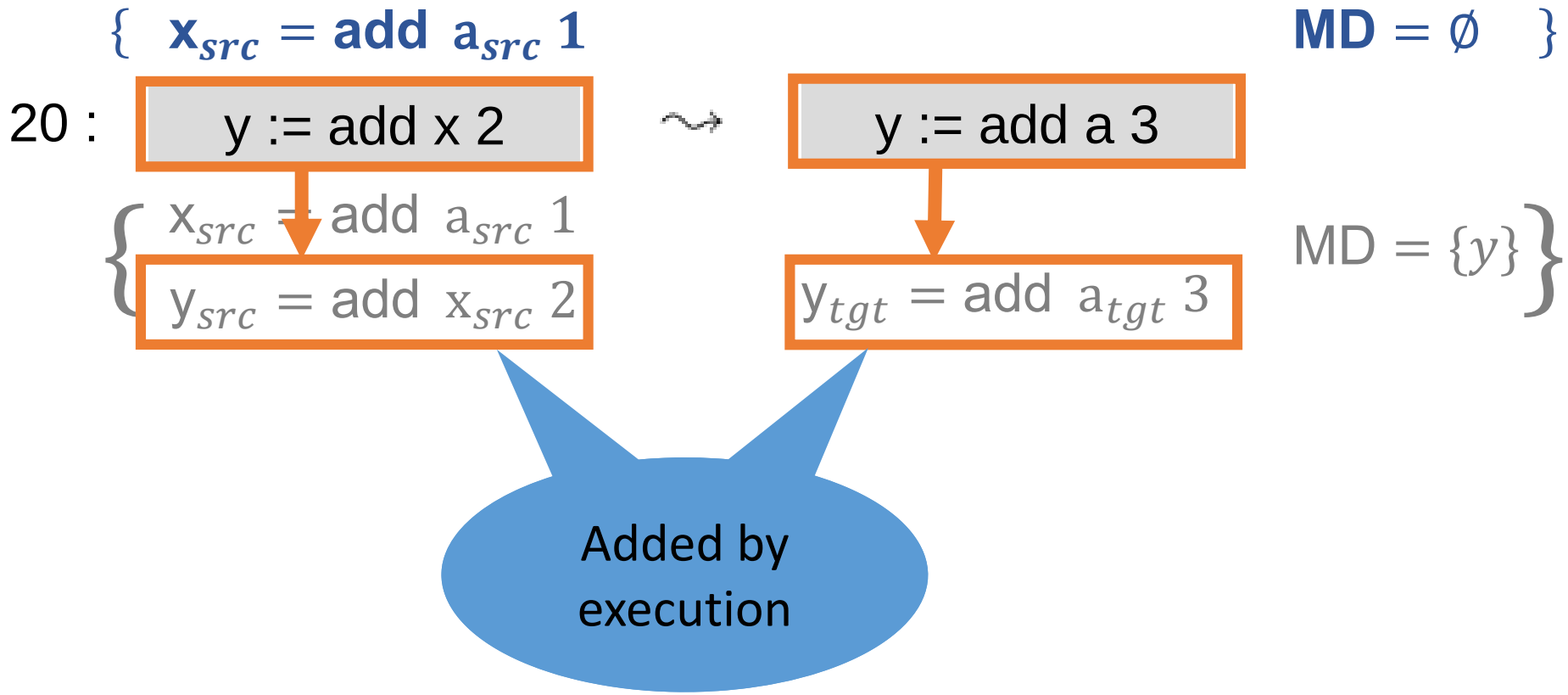
$\quad\ \ y_{src} = add\ x_{src}\ 2 \qquad\qquad y_{tgt} = add\ a_{tgt}\ 3$

$\{ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathbf{MD = \emptyset} \quad \}$

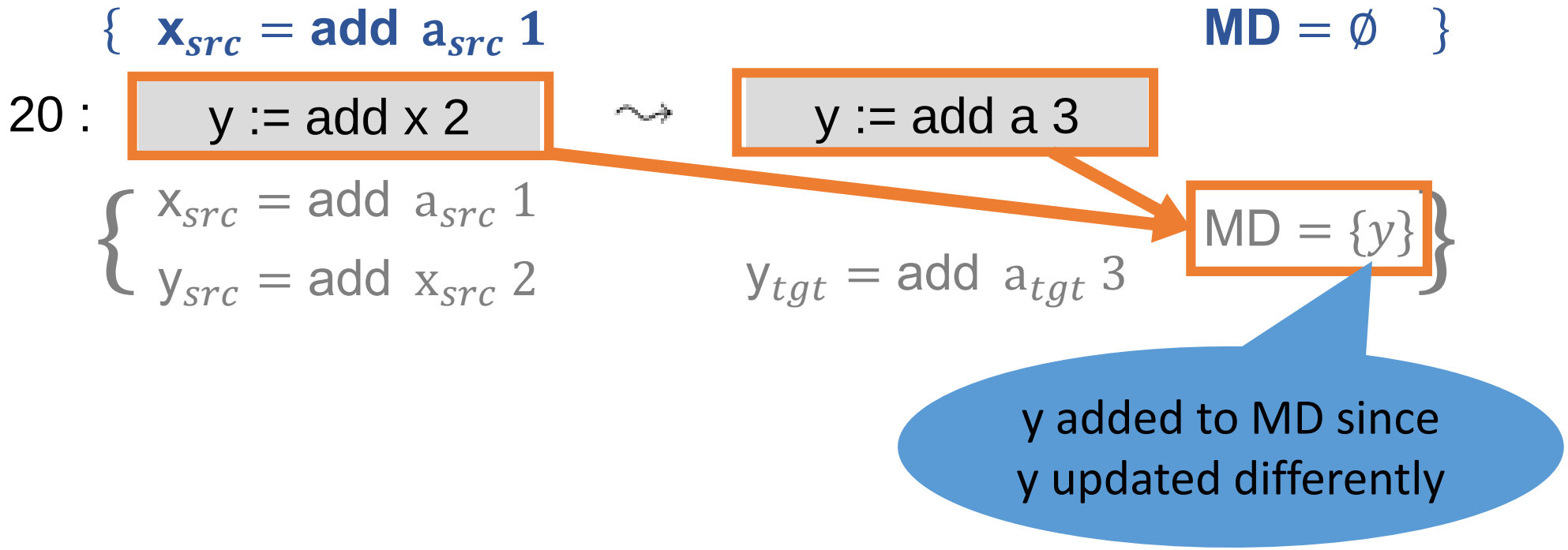
ERHL: A Logic for Optimization Validation



ERHL: A Logic for Optimization Validation



ERHL: A Logic for Optimization Validation



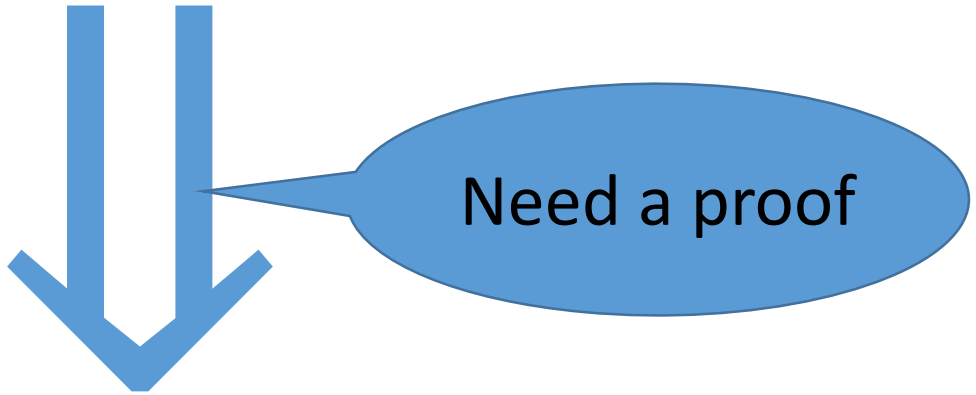
{ **MD** $= \emptyset$ }

ERHL: A Logic for Optimization Validation

$\{ \; \mathbf{x_{src} = add\ a_{src}\ 1} \qquad\qquad\qquad\qquad\qquad\qquad \mathbf{MD = \emptyset} \; \}$

20 : y := add x 2 $\rightsquigarrow$ y := add a 3

$$\left\{ \begin{array}{l} x_{src} = \text{add } a_{src}\ 1 \\ y_{src} = \text{add } x_{src}\ 2 \end{array} \right. \qquad y_{tgt} = \text{add } a_{tgt}\ 3 \qquad \left. MD = \{y\} \right\}$$



$\{ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathbf{MD = \emptyset} \; \}$

ERHL: A Logic for Optimization Validation

$\{\ \mathbf{x_{src} = add\ a_{src}\ 1}$ $\qquad\qquad\qquad\qquad\qquad\qquad$ $\mathbf{MD = \emptyset}\ \}$

20 : y := add x 2 $\rightsquigarrow$ y := add a 3

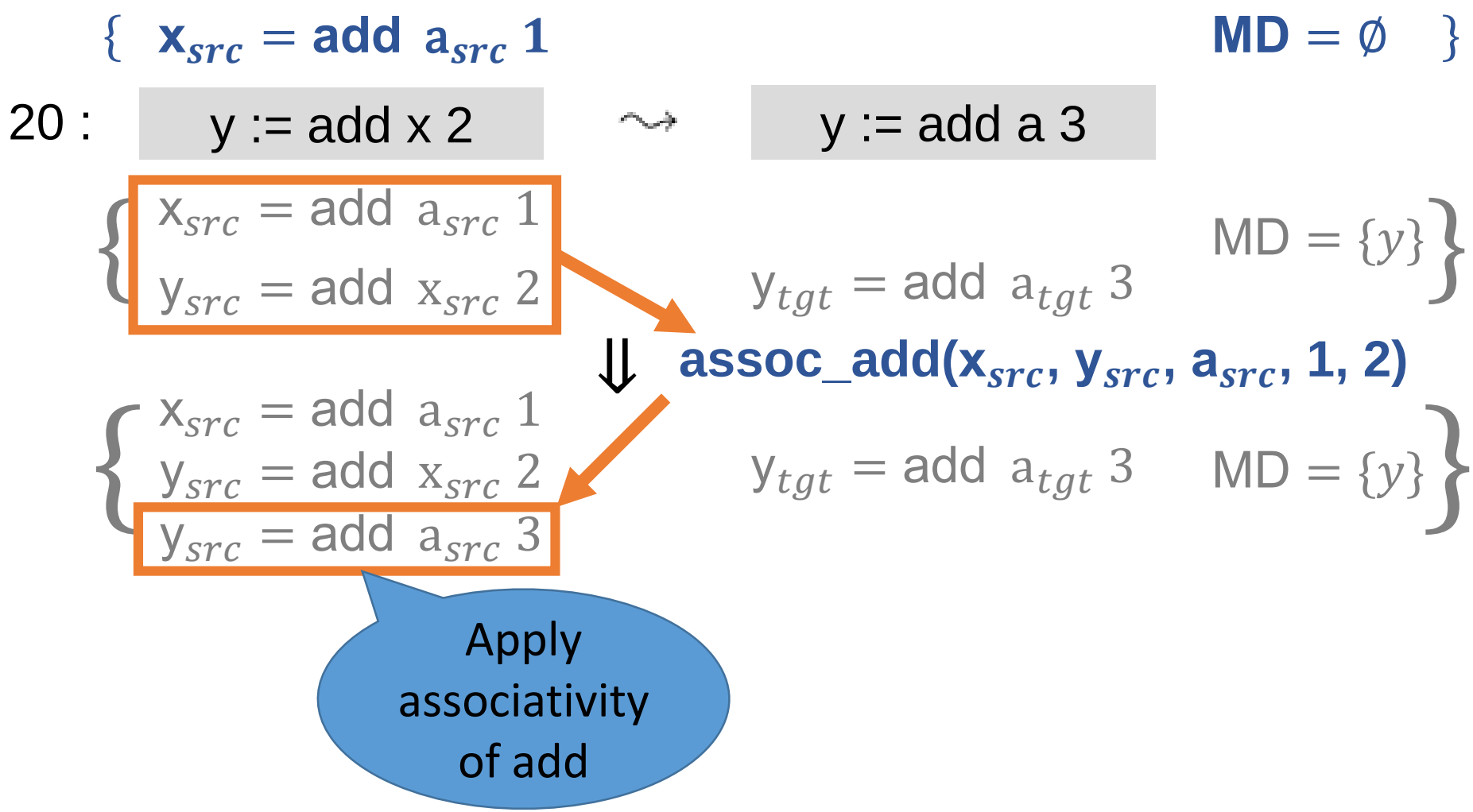
$$\left\{ \begin{array}{l} x_{src} = \text{add } a_{src}\ 1 \\ y_{src} = \text{add } x_{src}\ 2 \end{array} \right.$$ $\qquad$ $y_{tgt} = \text{add } a_{tgt}\ 3$ $\qquad$ $\left. MD = \{y\} \right\}$

$\Downarrow$ **assoc_add(x_{src}, y_{src}, a_{src}, 1, 2)**

$$\left\{ \begin{array}{l} x_{src} = \text{add } a_{src}\ 1 \\ y_{src} = \text{add } x_{src}\ 2 \\ y_{src} = \text{add } a_{src}\ 3 \end{array} \right.$$ $\qquad$ $y_{tgt} = \text{add } a_{tgt}\ 3$ $\qquad$ $\left. MD = \{y\} \right\}$

$\{$ $\qquad\qquad\qquad\qquad\qquad\qquad$ $\mathbf{MD = \emptyset}\ \}$

ERHL: A Logic for Optimization Validation



ERHL: A Logic for Optimization Validation

$\{\ \mathbf{x_{src} = add\ a_{src}\ 1}$ $\qquad\qquad\qquad\qquad\qquad\qquad$ $\mathbf{MD = \emptyset}\ \}$

20 : y := add x 2 $\rightsquigarrow$ y := add a 3

$$\left\{ \begin{array}{l} x_{src} = \text{add } a_{src}\ 1 \\ y_{src} = \text{add } x_{src}\ 2 \end{array} \right. \qquad y_{tgt} = \text{add } a_{tgt}\ 3 \qquad \left. MD = \{y\} \right\}$$

$\Downarrow$ **assoc_add(x_{src}, y_{src}, a_{src}, 1, 2)**

$$\left\{ \begin{array}{l} x_{src} = \text{add } a_{src}\ 1 \\ y_{src} = \text{add } x_{src}\ 2 \\ y_{src} = \text{add } a_{src}\ 3 \end{array} \right. \qquad y_{tgt} = \text{add } a_{tgt}\ 3 \qquad \left. MD = \{y\} \right\}$$

Propagated

$\{$ $\qquad\qquad\qquad\qquad\qquad\qquad$ $\mathbf{MD = \emptyset}\ \}$

ERHL: A Logic for Optimization Validation

$\{\ \mathbf{x_{src} = add\ a_{src}\ 1}\ \}$ $\quad\quad$ $\mathbf{MD = \emptyset}$ $\{$

20 : y := add x 2 $\quad\rightsquigarrow\quad$ y := add a 3

$$\left\{\begin{array}{l} x_{src} = \text{add } a_{src}\ 1 \\ y_{src} = \text{add } x_{src}\ 2 \end{array}\right.$$ $\quad\quad$ $y_{tgt} = \text{add } a_{tgt}\ 3$ $\quad\quad$ $\left.\text{MD} = \{y\}\right\}$

$\Downarrow$ **assoc_add(x_{src}, y_{src}, a_{src}, 1, 2)**

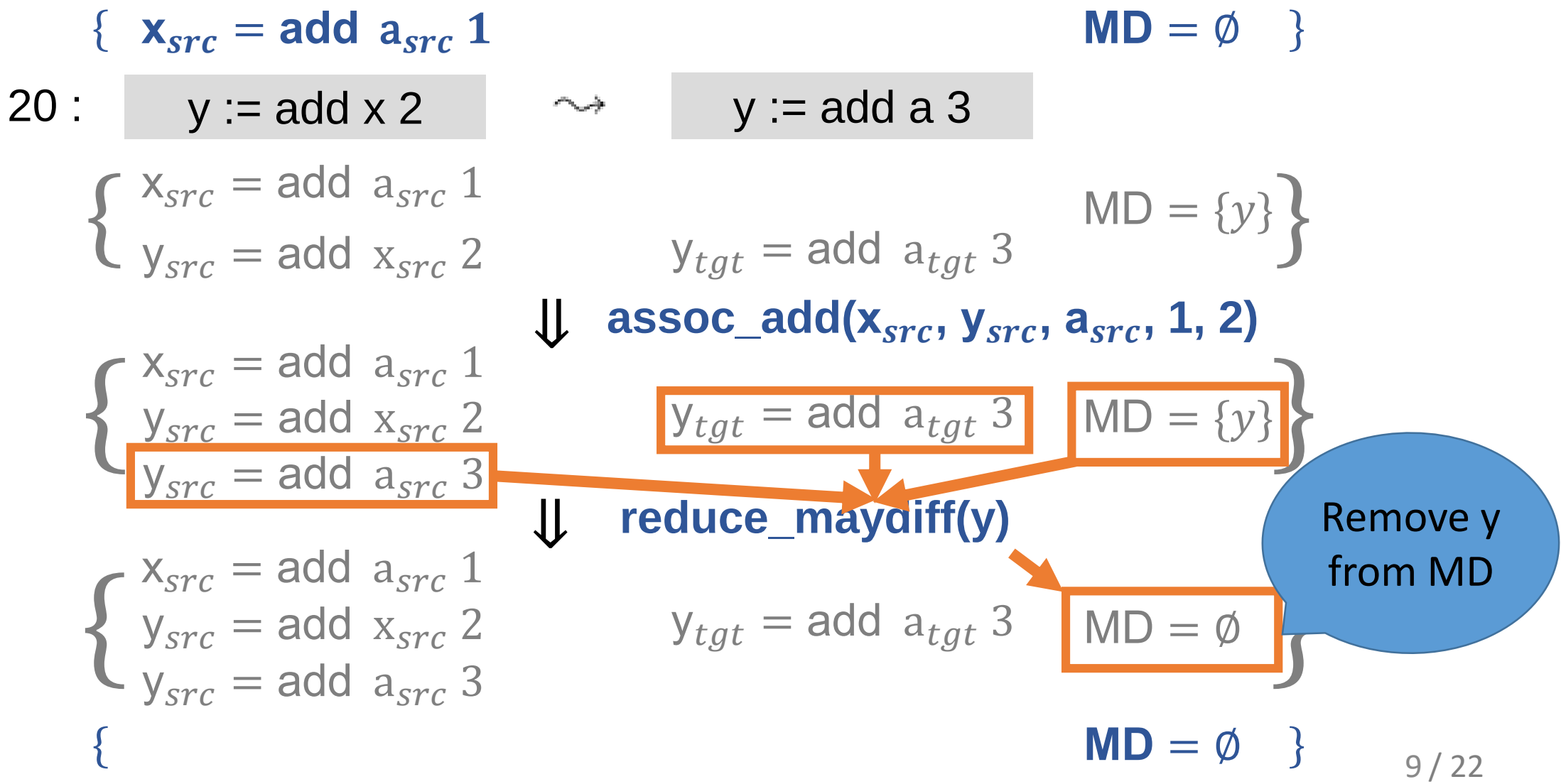
$$\left\{\begin{array}{l} x_{src} = \text{add } a_{src}\ 1 \\ y_{src} = \text{add } x_{src}\ 2 \\ y_{src} = \text{add } a_{src}\ 3 \end{array}\right.$$ $\quad\quad$ $y_{tgt} = \text{add } a_{tgt}\ 3$ $\quad\quad$ $\left.\text{MD} = \{y\}\right\}$

$\Downarrow$ **reduce_maydiff(y)**

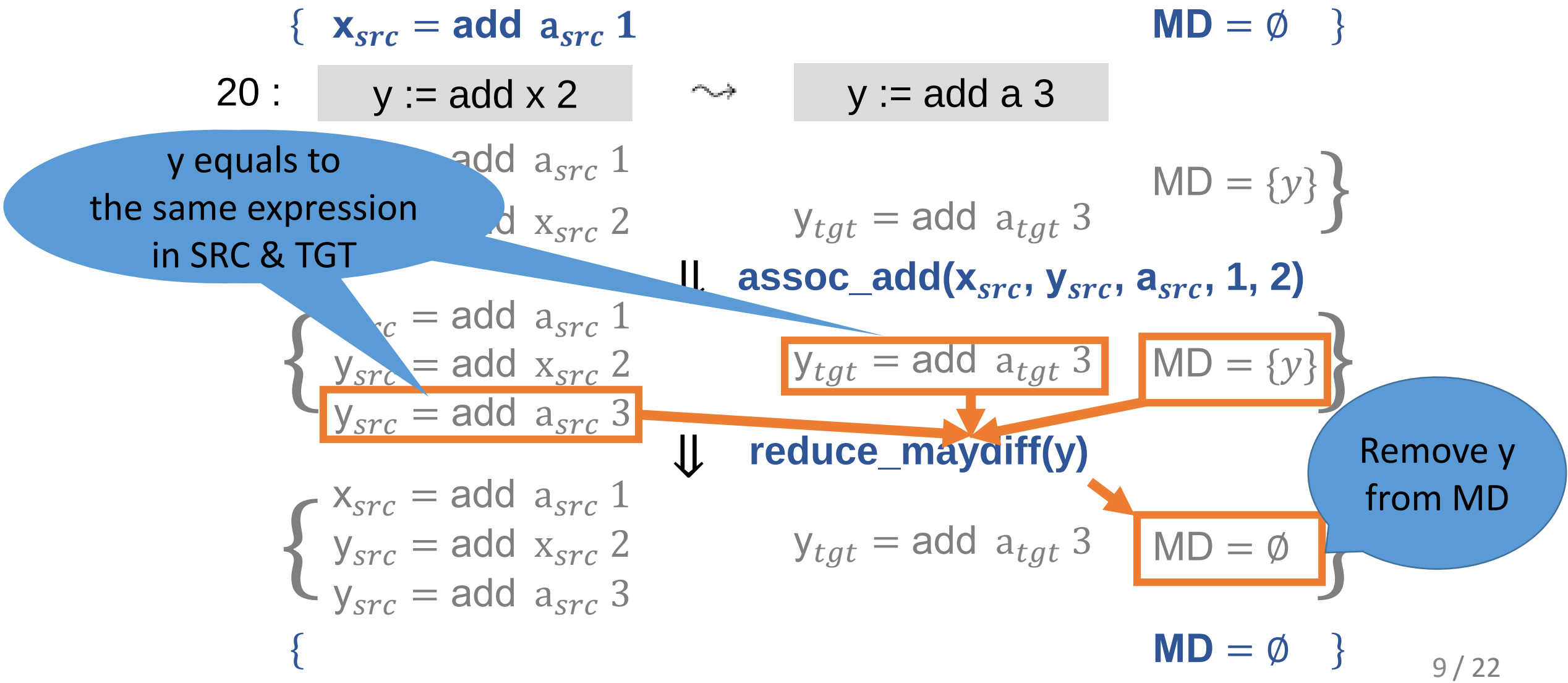
$$\left\{\begin{array}{l} x_{src} = \text{add } a_{src}\ 1 \\ y_{src} = \text{add } x_{src}\ 2 \\ y_{src} = \text{add } a_{src}\ 3 \end{array}\right.$$ $\quad\quad$ $y_{tgt} = \text{add } a_{tgt}\ 3$ $\quad\quad$ $\left.\text{MD} = \emptyset\right\}$

$\{$ $\quad\quad$ $\mathbf{MD = \emptyset}$ $\}$

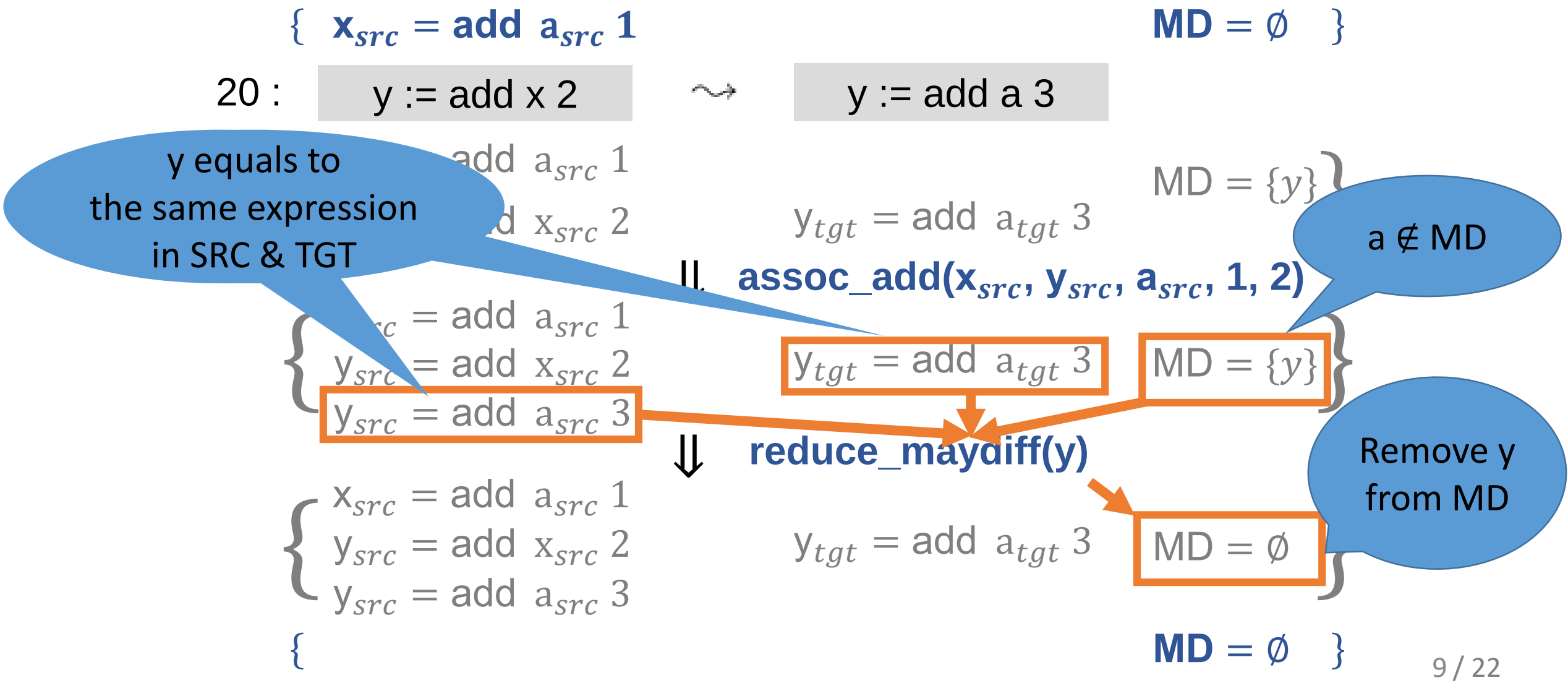
ERHL: A Logic for Optimization Validation



ERHL: A Logic for Optimization Validation



ERHL: A Logic for Optimization Validation



ERHL: A Logic for Optimization Validation

$\{\ \mathbf{x_{src} = add\ a_{src}\ 1}$ $\qquad\qquad\qquad\qquad\qquad\qquad$ $\mathbf{MD = \emptyset}\ \ \}$

20 : y := add x 2 $\rightsquigarrow$ y := add a 3

$$\left\{ \begin{array}{l} x_{src} = \text{add } a_{src}\ 1 \\ y_{src} = \text{add } x_{src}\ 2 \end{array} \right. \qquad y_{tgt} = \text{add } a_{tgt}\ 3 \qquad \left. \text{MD} = \{y\} \right\}$$

$\Downarrow$ **assoc_add(x_{src}, y_{src}, a_{src}, 1, 2)**

$$\left\{ \begin{array}{l} x_{src} = \text{add } a_{src}\ 1 \\ y_{src} = \text{add } x_{src}\ 2 \\ y_{src} = \text{add } a_{src}\ 3 \end{array} \right. \qquad y_{tgt} = \text{add } a_{tgt}\ 3 \qquad \left. \text{MD} = \{y\} \right\}$$

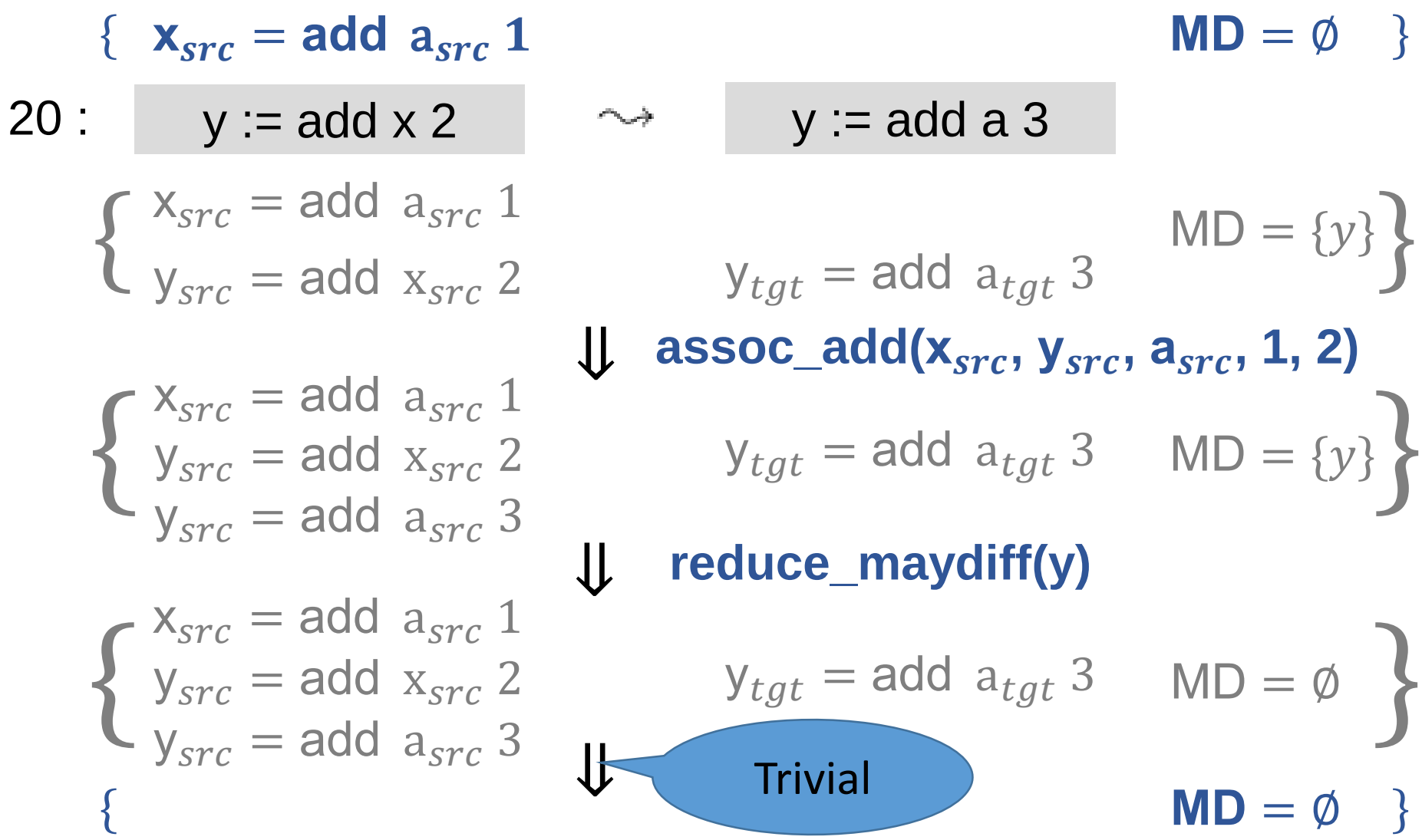
$\Downarrow$ **reduce_maydiff(y)**

$$\left\{ \begin{array}{l} x_{src} = \text{add } a_{src}\ 1 \\ y_{src} = \text{add } x_{src}\ 2 \\ y_{src} = \text{add } a_{src}\ 3 \end{array} \right. \qquad y_{tgt} = \text{add } a_{tgt}\ 3 \qquad \left. \text{MD} = \emptyset \right\}$$

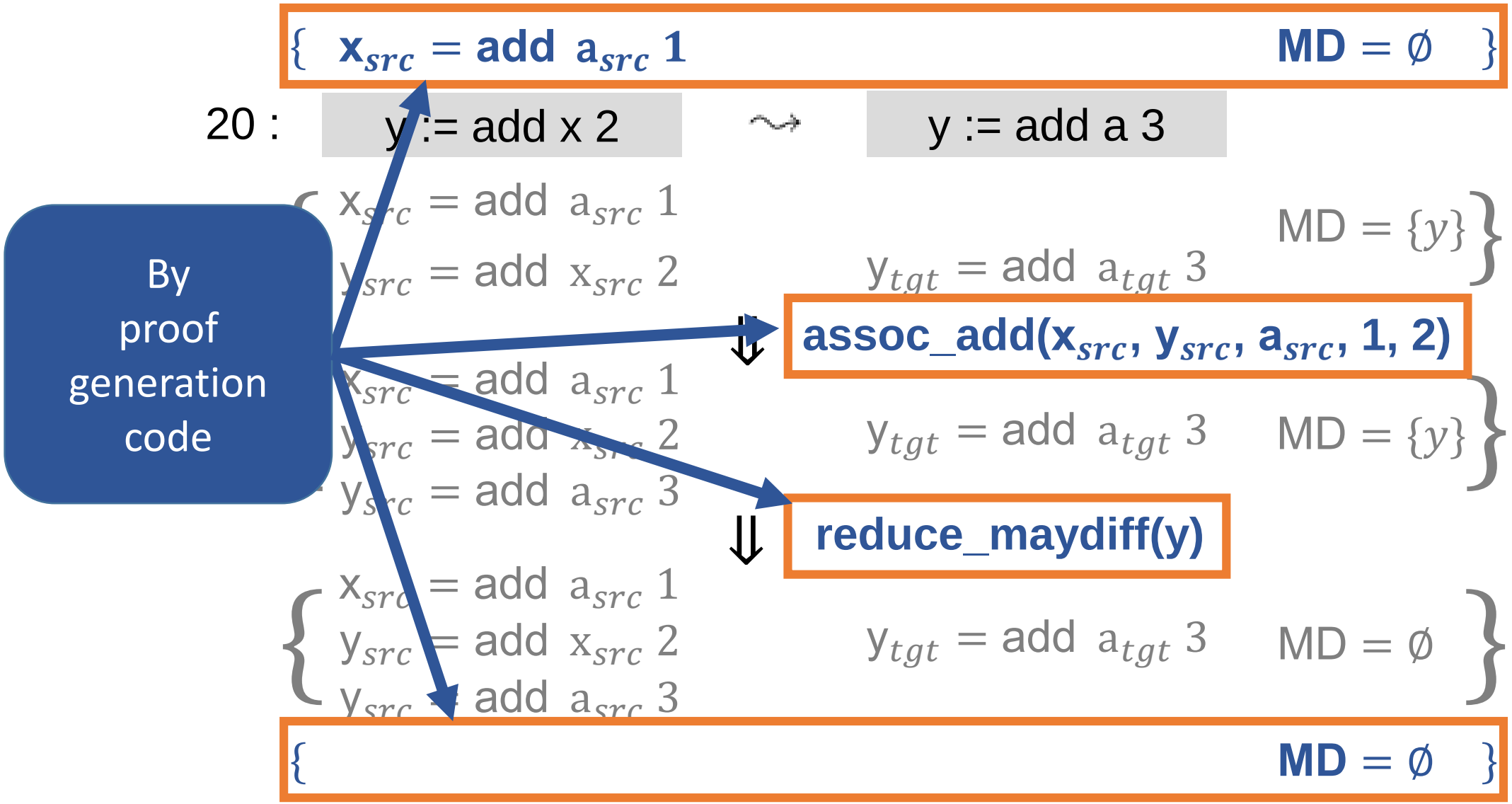
Propagated

$\mathbf{MD = \emptyset}\ \ \}$

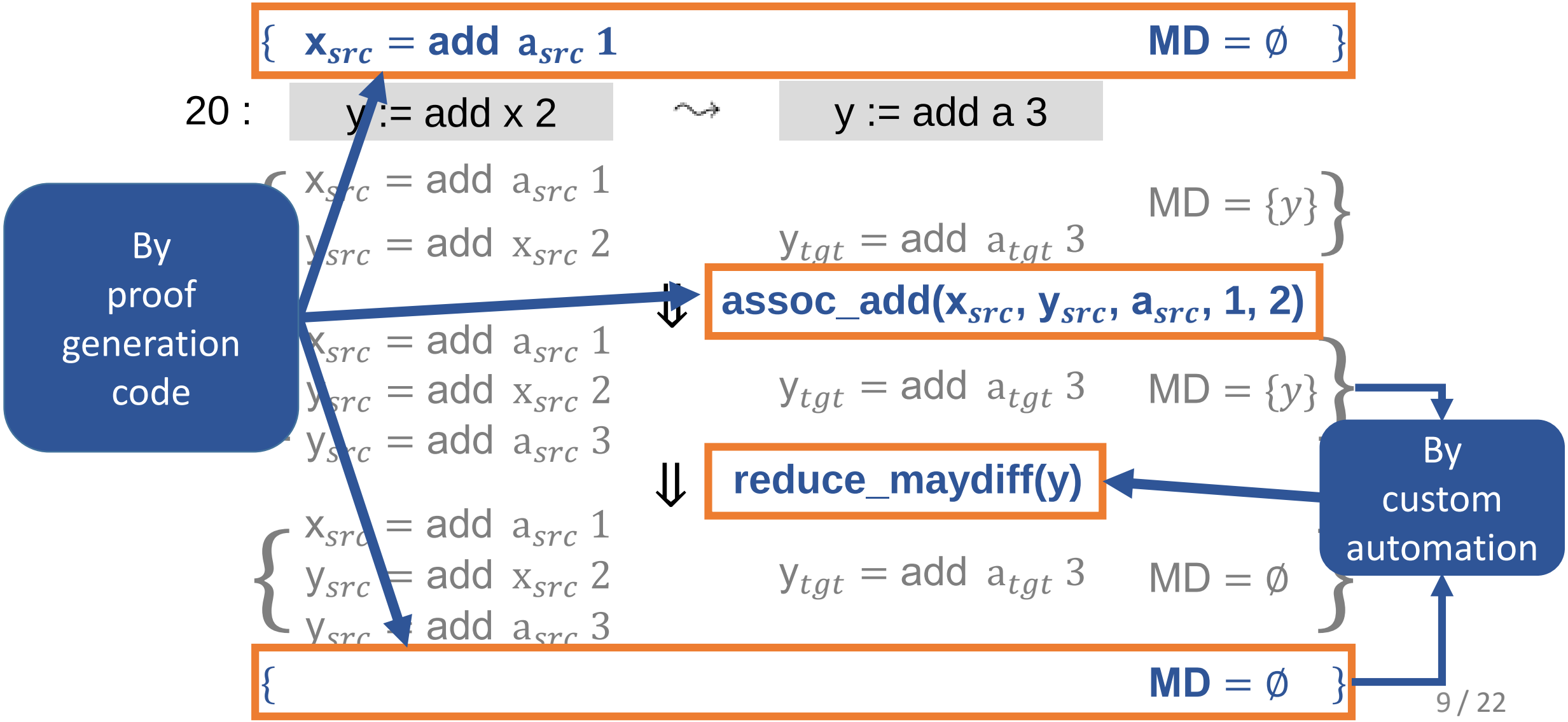
ERHL: A Logic for Optimization Validation



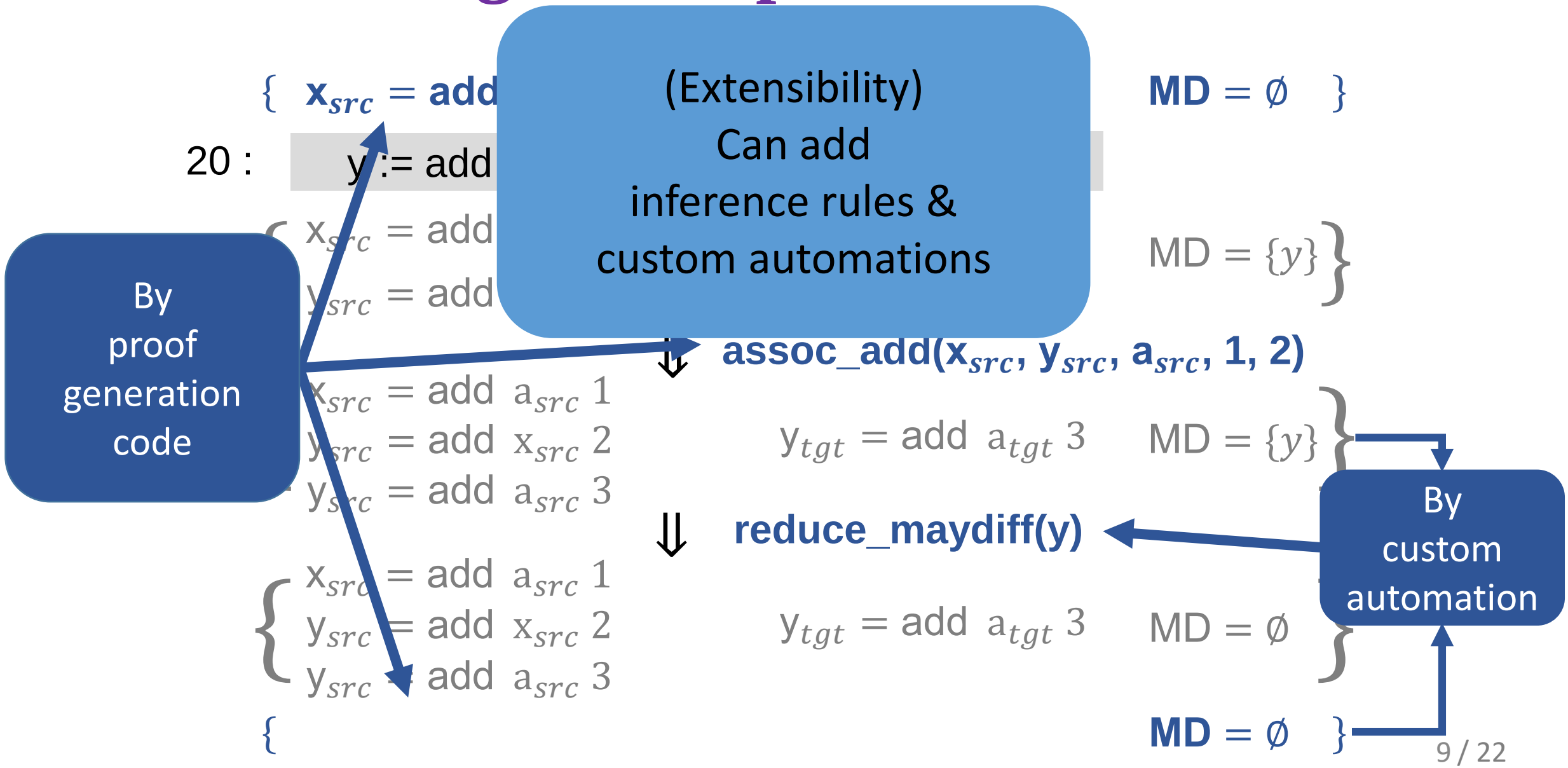
ERHL: A Logic for Optimization Validation



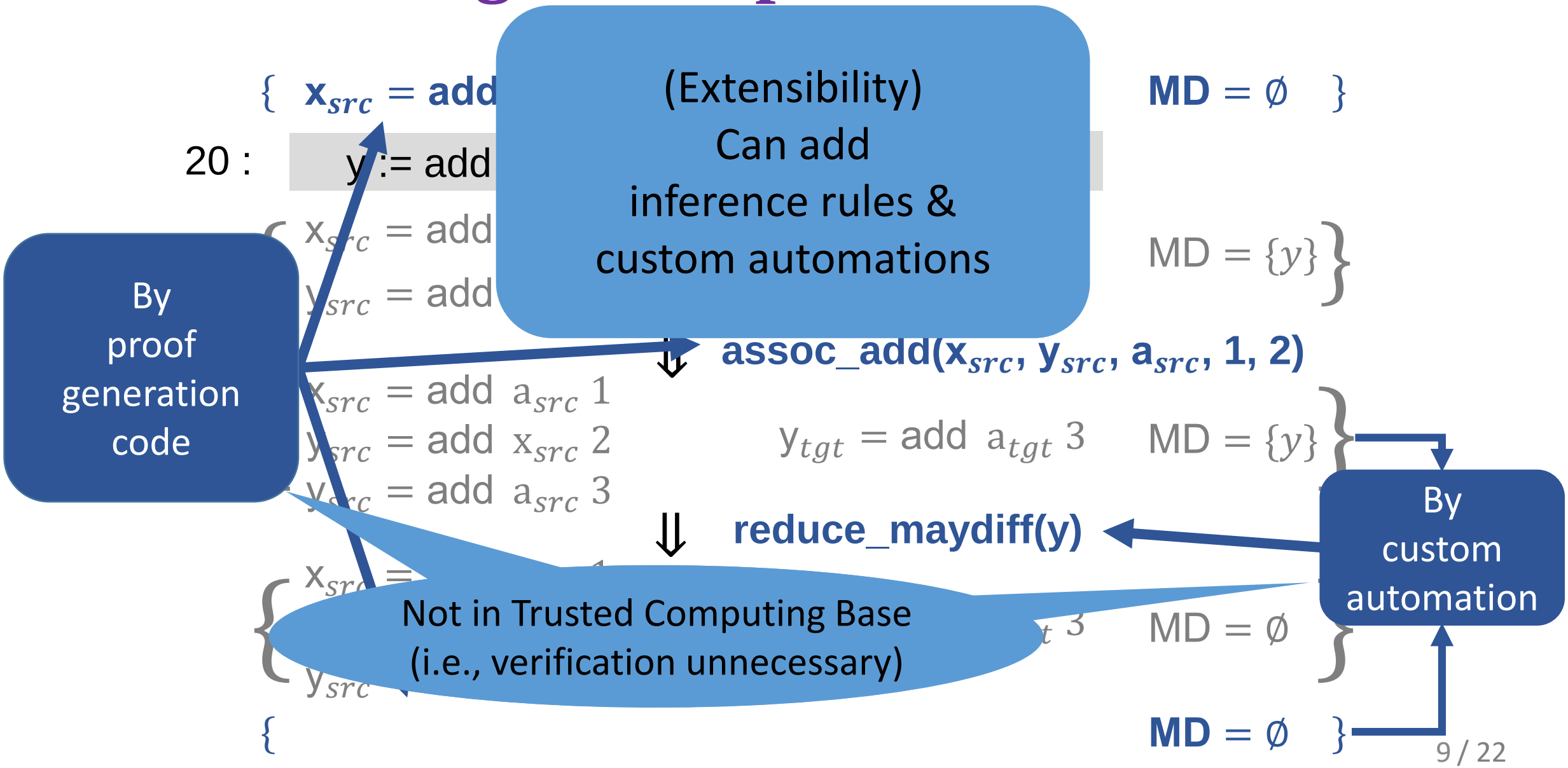
ERHL: A Logic for Optimization Validation



ERHL: A Logic for Optimization Validation



ERHL: A Logic for Optimization Validation



Proof Generation

Algorithm 1 AssocAdd(F: Function)

A1: **for** l_2: y := add (reg x) (const C_2) **in** F **do**

A2: **if** FindDef(F, x) **is** l_1: x := add (reg a) (const C_1) **then**

A3: C := Simplify(add C_1 C_2)

A4: ReplaceAt(F, l_2, y := add (reg a) (const C))

A5:

A6:

A7: **end if**

A8: **end for**

A9:

10 :	x := add a 1	⇝	x := add a 1
	⋮		⋮
20 :	y := add x 2	⇝	y := add a 3
21 :	foo(y)	⇝	foo(y)

Proof Generation

Algorithm 1 AssocAdd(F: Function)

A1: **for** l_2: y := add (reg x) (const C_2) **in** F **do**

A2: **if** FindDef(F, x) **is** l_1: x := add (reg a) (const C_1) **then**

A3: C := Simplify(add C_1 C_2)

A4: ReplaceAt(F, l_2, y := add (reg a) (const C))

A5:

A6:

A7: **end if**

A8: **end for**

A9:

10 :	x := add a 1	⇝	x := add a 1
	⋮		⋮
20 :	y := add x 2	⇝	y := add a 3
21 :	foo(y)	⇝	foo(y)

Proof Generation

Algorithm 1 AssocAdd(F: Function)

A1: **for** l_2: y := add (reg x) (const C_2) **in** F **do**

A2: **if** FindDef(F, x) **is** l_1: x := add (reg a) (const C_1) **then**

A3: C := Simplify(add C_1 C_2)

A4: ReplaceAt(F, l_2, y := add (reg a) (const C))

A5:

A6:

A7: **end if**

A8: **end for**

A9:

10 : x := add a 1 ⇝ x := add a 1

⋮ ⋮

20 : y := add x 2 ⇝ y := add a 3

21 : foo(y) ⇝ foo(y)

Proof Generation

$C_1 = 1$
$C_2 = 2$
$\Downarrow$
$C = 3$

Algorithm 1 AssocAdd(F: Function)

A1: for l_2: y := add (reg x) (const C_2) **in** F **do**
A2: **if** FindDef(F, x) **is** l_1: x := add (reg a) (const C_1) **then**
A3: C := Simplify(add C_1 C_2)
A4: ReplaceAt(F, l_2, y := add (reg a) (const C))
A5:
A6:
A7: **end if**
A8: **end for**
A9:

10:	x := add a 1	⇝	x := add a 1
⋮	⋮		⋮
20:	y := add x 2	⇝	y := add a 3
21 :	foo(y)	⇝	foo(y)

Proof Generation

$C_1 = 1$
$C_2 = 2$
$\Downarrow$
$C = 3$

Algorithm 1 AssocAdd(F: Function)

A1: **for** l_2: y := add (reg x) (const C_2) **in** F **do**
A2: **if** FindDef(F, x) **is** l_1: x := add (reg a) (const C_1) **then**
A3: C := Simplify(add C_1 C_2)
A4: ReplaceAt(F, l_2, y := add (reg a) (const C))
A5:
A6:
A7: **end if**
A8: **end for**
A9:

10 : x := add a 1 ⇝ x := add a 1

⋮ ⋮

20 : y := add x 2 ⇝ y := add a 3

21 : foo(y) ⇝ foo(y)

Proof Generation

Algorithm 1 AssocAdd(F: Function)

A1: **for** l_2: y := add (reg x) (const C_2) **in** F **do**

A2: **if** FindDef(F, x) **is** l_1: x := add (reg a) (const C_1) **then**

A3: C := Simplify(add C_1 C_2)

A4: ReplaceAt(F, l_2, y := add (reg a) (const C))

A5:

A6:

A7: **end if**

A8: **end for**

A9:

{ **MD** = ∅ }

10: x := add a 1 ⇝ x := add a 1

{ **MD** = ∅ }

⋮ ⋮

{ **MD** = ∅ }

20: y := add x 2 ⇝ y := add a 3

{ **MD** = ∅ }

21 : foo(y) ⇝ foo(y)

{ **MD** = ∅ }

Proof Generation

Algorithm 1 AssocAdd(F: Function)

A1: **for** l_2: y := add (reg x) (const C_2) **in** F **do**

A2: **if** FindDef(F, x) **is** l_1: x := add (reg a) (const C_1) **then**

A3: C := Simplify(add C_1 C_2)

A4: ReplaceAt(F, l_2, y := add (reg a) (const C))

A5: Assn(x_{src} = add a_{src} C_1, l_1, l_2)

A6:

A7: **end if**

A8: **end for**

A9:

{ **MD** = ∅ }

10: x := add a 1 ⇝ x := add a 1

{ $\mathbf{x}_{src}$ = **add** $\mathbf{a}_{src}$ **1** **MD** = ∅ }

⋮ ⋮

{ $\mathbf{x}_{src}$ = **add** $\mathbf{a}_{src}$ **1** **MD** = ∅ }

20: y := add x 2 ⇝ y := add a 3

{ **MD** = ∅ }

21 : foo(y) ⇝ foo(y)

{ **MD** = ∅ }

Proof Generation

Algorithm 1 AssocAdd(F: Function)

A1: **for** l_2: y := add (reg x) (const C_2) **in** F **do**

A2: **if** FindDef(F, x) **is** l_1: x := add (reg a) (const C_1) **then**

A3: C := Simplify(add C_1 C_2)

A4: ReplaceAt(F, l_2, y := add (reg a) (const C))

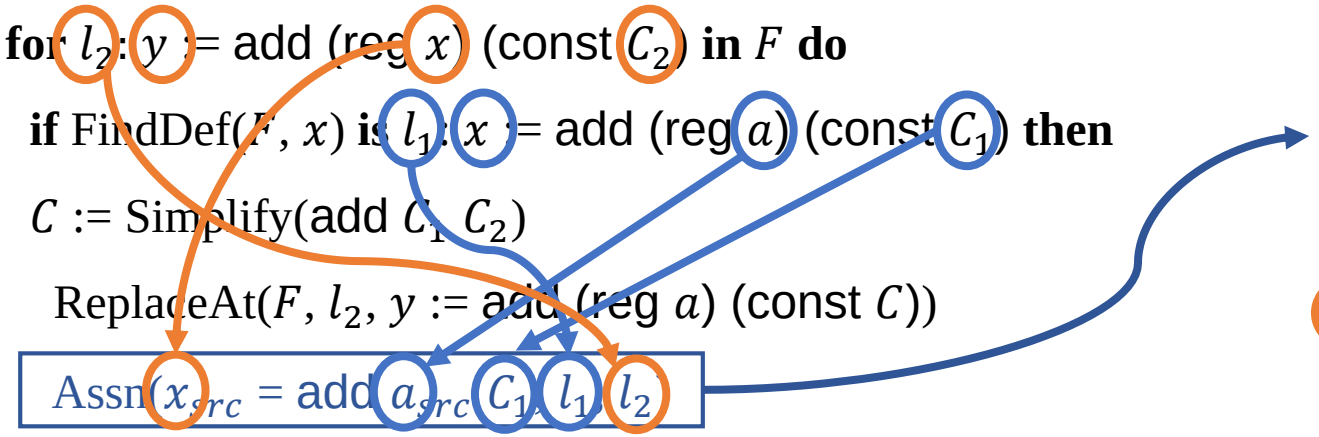
A5: Assn(x_{src} = add a_{src} C_1 l_1 l_2)

A6:

A7: **end if**

A8: **end for**

A9:



{ **MD** = ∅ }

10: x := add a 1 ⇝ x := add a 1

{ $\mathbf{x}_{src}$ = **add** $\mathbf{a}_{src}$ **1** **MD** = ∅ }

⋮ ⋮

{ $\mathbf{x}_{src}$ = **add** $\mathbf{a}_{src}$ **1** **MD** = ∅ }

20: y := add x 2 ⇝ y := add a 3

{ **MD** = ∅ }

21 : foo(y) ⇝ foo(y)

{ **MD** = ∅ }

Proof Generation

Algorithm 1 AssocAdd(F: Function)

A1: **for** l_2: y := add (reg x) (const C_2) **in** F **do**

A2: **if** FindDef(F, x) **is** l_1: x := add (reg a) (const C_1) **then**

A3: C := Simplify(add C_1 C_2)

A4: ReplaceAt(F, l_2, y := add (reg a) (const C))

A5: Assn(x_{src} = add a_{src} C_1, l_1, l_2)

A6: Inf(assoc_add(x_{src}, y_{src}, a_{src}, C_1, C_2), l_2)

A7: **end if**

A8: **end for**

A9:

{ **MD** = ∅ }

10: x := add a 1 ⇝ x := add a 1

{ $\mathbf{x}_{src}$ = **add** $\mathbf{a}_{src}$ **1** **MD** = ∅ }

⋮ ⋮

{ $\mathbf{x}_{src}$ = **add** $\mathbf{a}_{src}$ **1** **MD** = ∅ }

20: y := add x 2 ⇝ y := add a 3

assoc_add($\mathbf{x}_{src}$, $\mathbf{y}_{src}$, $\mathbf{a}_{src}$, 1, 2)

{ **MD** = ∅ }

21 : foo(y) ⇝ foo(y)

{ **MD** = ∅ }

Proof Generation

Algorithm 1 AssocAdd(F: Function)

A1: **for** l_2: y := add (reg x) (const C_2) **in** F **do**

A2: **if** FindDef(F, x) **is** l_1: x := add (reg a) (const C_1) **then**

A3: C := Simplify(add C_1 C_2)

A4: ReplaceAt(F, l_2, y := add (reg a) (const C))

A5: Assn(x_{src} = add a_{src} C_1, l_1, l_2)

A6: Inf(assoc_add(x_{src}, y_{src}, a_{src}, C_1, C_2), l_2)

A7: **end if**

A8: **end for**

A9:

{ **MD = ∅** }

10: x := add a 1 ⇝ x := add a 1

{ $\mathbf{x_{src}}$ = **add** $\mathbf{a_{src}}$ **1** **MD = ∅** }

⋮ ⋮

{ $\mathbf{x_{src}}$ = **add** $\mathbf{a_{src}}$ **1** **MD = ∅** }

20: y := add x 2 ⇝ y := add a 3

assoc_add($\mathbf{x_{src}}$, $\mathbf{y_{src}}$, $\mathbf{a_{src}}$, 1, 2)

{ **MD = ∅** }

21 : foo(y) ⇝ foo(y)

{ **MD = ∅** }

Proof Generation

Algorithm 1 AssocAdd(F: Function)

A1: **for** l_2: y := add (reg x) (const C_2) **in** F **do**

A2: **if** FindDef(F, x) **is** l_1: x := add (reg a) (const C_1) **then**

A3: C := Simplify(add C_1 C_2)

A4: ReplaceAt(F, l_2, y := add (reg a) (const C))

A5: Assn(x_{src} = add a_{src} C_1, l_1, l_2)

A6: Inf(assoc_add(x_{src}, y_{src}, a_{src}, C_1, C_2), l_2)

A7: **end if**

A8: **end for**

A9: EnableAuto(reduce_maydiff)

{ **MD** = ∅ }

10: x := add a 1 ⇝ x := add a 1

{ $\mathbf{x}_{src}$ = **add** $\mathbf{a}_{src}$ **1** **MD** = ∅ }

⋮ ⋮

{ $\mathbf{x}_{src}$ = **add** $\mathbf{a}_{src}$ **1** **MD** = ∅ }

20: y := add x 2 ⇝ y := add a 3

assoc_add($\mathbf{x}_{src}$, $\mathbf{y}_{src}$, $\mathbf{a}_{src}$, 1, 2)

reduce_maydiff(y)

{ **MD** = ∅ }

21 : foo(y) ⇝ foo(y)

{ **MD** = ∅ }

Proof Generation

Algorithm 1 AssocAdd(F: Function)

A1: **for** l_2: y := add (reg x) (const C_2) **in** F **do**

A2: **if** FindDef(F, x) **is** l_1: x := add (reg a) (const C_1) **then**

A3: C := Simplify(add C_1 C_2)

A4: ReplaceAt(F, l_2, y := add (reg a) (const C))

A5: Assn(x_{src} = add a_{src} C_1, l_1, l_2)

A6: Inf(assoc_add(x_{src}, y_{src}, a_{src}, C_1, C_2), l_2)

A7: **end if**

A8: **end for**

A9: EnableAuto(reduce_maydiff)

Extra code for Crellvm

{ **MD** = ∅ }

10 : x := add a 1 ⇝ x := add a 1

{ $\mathbf{x_{src}}$ = **add** $\mathbf{a_{src}}$ **1** **MD** = ∅ }

⋮ ⋮

{ $\mathbf{x_{src}}$ = **add** $\mathbf{a_{src}}$ **1** **MD** = ∅ }

= add x 2 ⇝ y := add a 3

assoc_add($\mathbf{x_{src}}$, $\mathbf{y_{src}}$, $\mathbf{a_{src}}$, 1, 2)

reduce_maydiff(y)

{ **MD** = ∅ }

21 : foo(y) ⇝ foo(y)

{ **MD** = ∅ }

General Relational Properties in ERHL

- From the register promotion optimization

$$\left\{ \quad *\mathrm{p}_{src} = \mathrm{r}_{tgt} \qquad\qquad \mathbf{MD} = \{\mathrm{p}, \mathrm{r}\} \quad \right\}$$

General Relational Properties in ERHL

- From the register promotion optimization

$$\{\ *\mathrm{p}_{src} = \mathrm{r}_{tgt} \quad \mathbf{MD} = \{\mathrm{p}, \mathrm{r}\}\ \}$$

Not allowed in ERHL

General Relational Properties in ERHL

- From the register promotion optimization

$$\{\ *\mathrm{p}_{src} = \mathrm{r}_{tgt} \qquad \mathbf{MD} = \{\mathrm{p}, \mathrm{r}\}\ \}$$

Encoded as

$$\{\ *\mathrm{p}_{src} = \hat{\mathrm{p}}_{src} \qquad \hat{\mathrm{p}}_{tgt} = \mathrm{r}_{tgt} \qquad \mathbf{MD} = \{\mathrm{p}, \mathrm{r}\}\ \}$$

General Relational Properties in ERHL

- From the register promotion optimization

$$\{\ *p_{src} = r_{tgt}\ \ \mathbf{MD} = \{p, r\}\ \}$$

Encoded as

$$\{\ *p_{src} = \hat{p}_{src}\ \ \hat{p}_{tgt} = r_{tgt}\ \ \mathbf{MD} = \{p, r\}\ \}$$

$$*p_{src} = \hat{p}_{src}$$

General Relational Properties in ERHL

- From the register promotion optimization

$$\{\ *p_{src} = r_{tgt} \qquad MD = \{p, r\}\ \}$$

Encoded as

$$\{\ *p_{src} = \hat{p}_{src} \qquad \hat{p}_{tgt} = r_{tgt} \qquad MD = \{p, r\}\ \}$$

$\hat{p} \notin MD$

$$*p_{src} = \hat{p}_{src} = \hat{p}_{tgt}$$

General Relational Properties in ERHL

- From the register promotion optimization

$$\{\ *p_{src} = r_{tgt}\ \ \ \mathbf{MD} = \{p, r\}\ \}$$

Encoded as

$$\{\ *p_{src} = \hat{p}_{src}\ \ \ \hat{p}_{tgt} = r_{tgt}\ \ \ \mathbf{MD} = \{p, r\}\ \}$$

$$*p_{src} = \hat{p}_{src} = \hat{p}_{tgt} = r_{tgt}$$

General Relational Properties in ERHL

- From the register promotion optimization

$$\{\ *\mathrm{p}_{src} = \mathrm{r}_{tgt} \qquad \mathbf{MD} = \{\mathrm{p}, \mathrm{r}\}\ \}$$

Encoded as

$$\{\ *\mathrm{p}_{src} = \hat{\mathrm{p}}_{src} \qquad \hat{\mathrm{p}}_{tgt} = \mathrm{r}_{tgt} \qquad \mathbf{MD} = \{\mathrm{p}, \mathrm{r}\}\ \}$$

(Ghost registers)
- existentially quantified
- introduced by inf rule

$$*\mathrm{p}_{src} = \hat{\mathrm{p}}_{src} = \hat{\mathrm{p}}_{tgt} = \mathrm{r}_{tgt}$$

Implementation & Results

Implementation: Proof Checker

- Implemented & Verified soundness in Coq
  - Used formal LLVM semantics from Vellvm
  - Proved semantics preservation using CompCert's memory injection
- Installed 221 inference rules
  - 9 main rules (verified)
  - 212 arithmetic & special rules for instcombine (unverifed)

	Implementation	**Verification**	**Total**
Proof Checker (SLOC)	2,987	18,934	21,921

Implementation: Proof Checker

- Implemented & Verified soundness in Coq
  - Used formal LLVM semantics from Vellvm
  - Proved semantics preservation using CompC… …on
- Installed 221 inference rules
  - 9 main rules (verified)
    - transitivity
    - reduce_maydiff
    - intro_ghost
    - …
  - 212 arithmetic & special rules for instcombine (unverifed)

	Implementation	Verification	Total
Proof Checker (SLOC)	2,987	18,934	21,921

Implementation: Proof Generation

- Covered 4 passes in LLVM 3.7.1
  - `mem2reg`: register promotion algorithm
  - `gvn`: GVN-PRE algorithm
  - `licm`: loop-invariant code motion algorithm (partially covered)
  - `instcombine`: 158 peephole optimizations among >1000 ones

	`mem2reg`	`gvn`	`licm`	`instcombine`
Compiler (Covered SLOC)	568	1,092	706	702
Proof Generation (SLOC)	213	440	286	1,357

Experiment Results

	LOC	Validations	Fail	Not Supported	Success
SPEC CINT 2006	1.0M	390K	69	31K (8%)	359K (92%)
LLVM Nightly	1.4M	907K	69	361K (40%)	546K (60%)
Open-Source	2.9M	908K	325	180K (20%)	728K (80%)
Random (Csmith)	1.6M	98K	1	12K (12%)	86K (88%)
Total	6.9M	2303K	464	584K (25%)	1719K (75%)

* Open-Source: Sendmail, Emacs, Python, Gimp, Ghostscript

Experiment Results

	LOC	Validations	Fail	Not Supported	Success
SPEC CINT 2006	1.0M	390K	69	31K (8%)	359K (92%)
LLVM Nightly	1.4M	907K	69	361K (40%)	546K (60%)
Open-Source	2.9M	908K	325	180K (20%)	728K (80%)
Random (Csmith)	1.6M	98K	1	12K (12%)	86K (88%)
Total	6.9M	2303K	464	584K (25%)	1719K (75%)

All due to 4 compiler bugs

* Open-Source: Sendmail, Emacs, Python, Gimp, Ghostscript

Experiment Results

	LOC	Validations	Fail	Not Supported	Success
SPEC CINT 2006	1.0M	390K	69	31K (8%)	359K (92%)
LLVM Nightly	1.4M	907K	69	361K (40%)	546K (60%)
Open-Source	2.9M	908K	325	180K (20%)	728K (80%)
Random (Csmith)	1.6M	98K	1	12K (12%)	86K (88%)
Total	6.9M	2303K	464	584K (25%)	1719K (75%)

All due to 4 compiler bugs

Lack of language formalization (e.g., vector operations)

* Open-Source: Sendmail, Emacs, Python, Gimp, Ghostscript

Execution Times

	Compilation Phase	Validation Phase
SPEC CINT2006	51	141K
LLVM Nightly	82	126K
Open-Source	125	175K
Total (sec.)	258	442K

Execution Times

	Compilation Phase	Validation Phase
SPEC CINT2006	51	141K
LLVM Nightly	82	126K
Open-Source	125	175K
Total (sec.)	258	442K

4 min : 123 hour
= 1 : 1700

Execution Times

	Compilation Phase	Validation Phase
SPEC CINT2006	51	141K
LLVM Nightly	82	126K
Open-Source	125	175K
Total (sec.)	258	442K

4 min : 123 hour
= 1 : 1700

But, embarrassingly parallel
(~3 hours in wall clock,
in 96 threads)

Summary

- Developed a verified credible compilation framework for LLVM
- Covered 3 major and >100 peephole optimizations of LLVM
- Discovered 4 long-standing bugs in LLVM
  - 2 in mem2reg
  - 2 in gvn

What else is in the paper?

- Reasoning about cyclic control flows
- Reasoning about Memory properties
- Details of `mem2reg` and `gvn` validation
- Porting to LLVM 5.0.1

Future Work

- Formalizing LLVM IR features in Vellvm
  - Vectors, Readonly, etc
  - Integer-pointer casts
  - Undef & poison values
- Supporting complex analyses
  - General alias analysis, division-by-zero, etc
- Supporting CFG-changing optimizations
  - Loop unrolling, etc