

# 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.

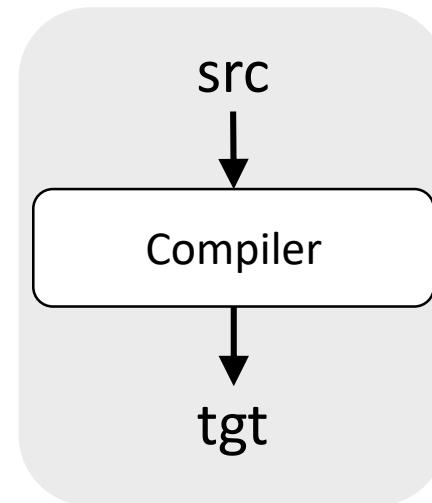
# Reliability of Production Compilers

- 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

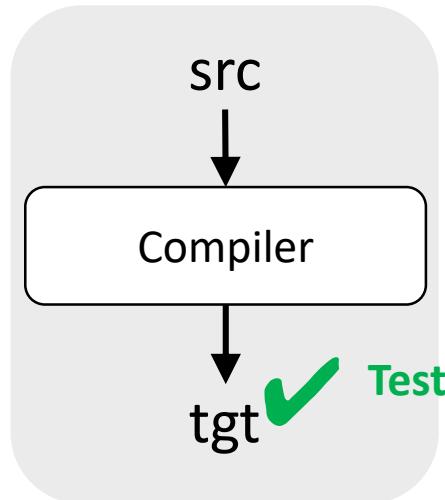
- Stable in most common cases
  - Unstable in corner cases
    - Csmith: 79 from GCC / 202 from LLVM
    - EMI: 79 from GCC / 68 from LLVM
  - Problematic in practice
    - Low-level systems code
- ➔ Goal: Improving reliability in corner cases

# Approaches to Improving Reliability



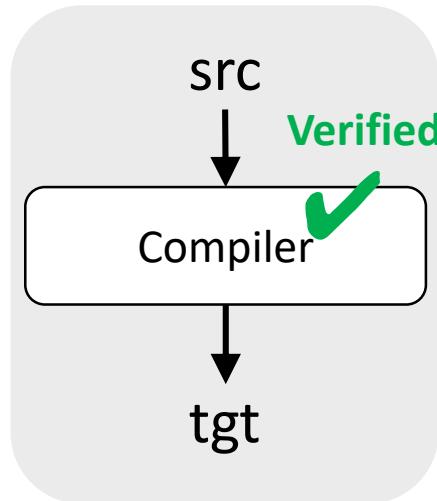
# Approaches to Improving Reliability

- (Random) Testing
  - Cannot guarantee high reliability



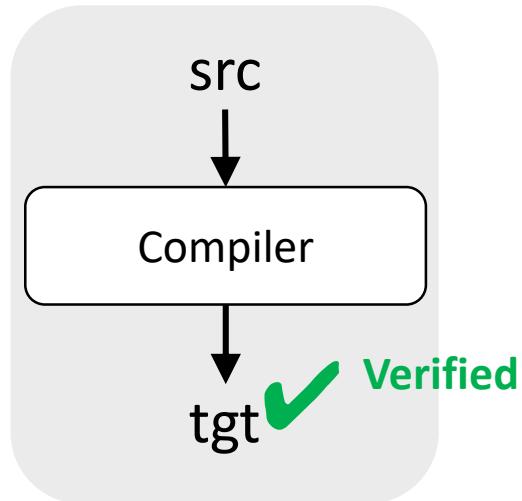
# Approaches to Improving Reliability

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



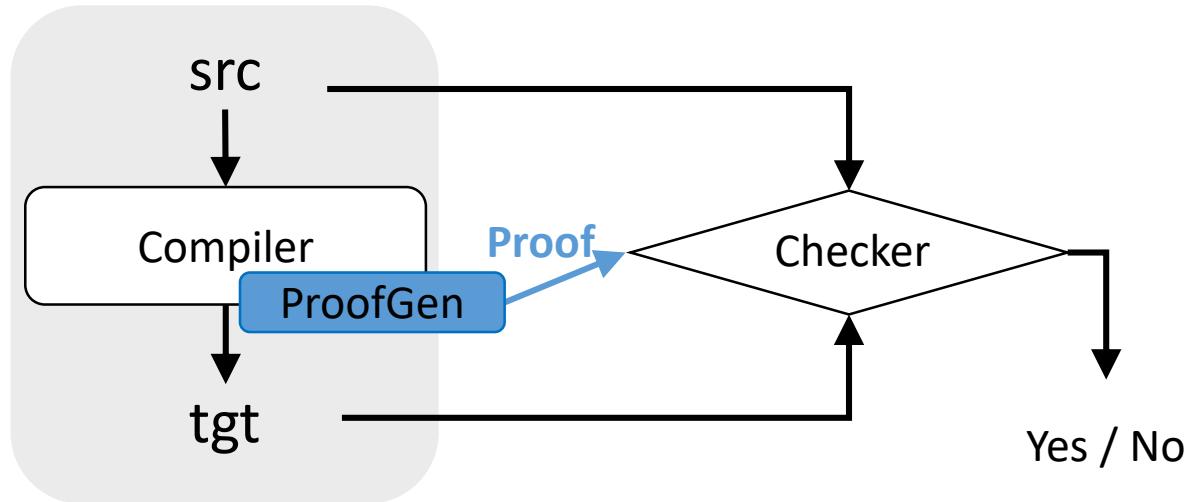
# Approaches to Improving Reliability

- (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



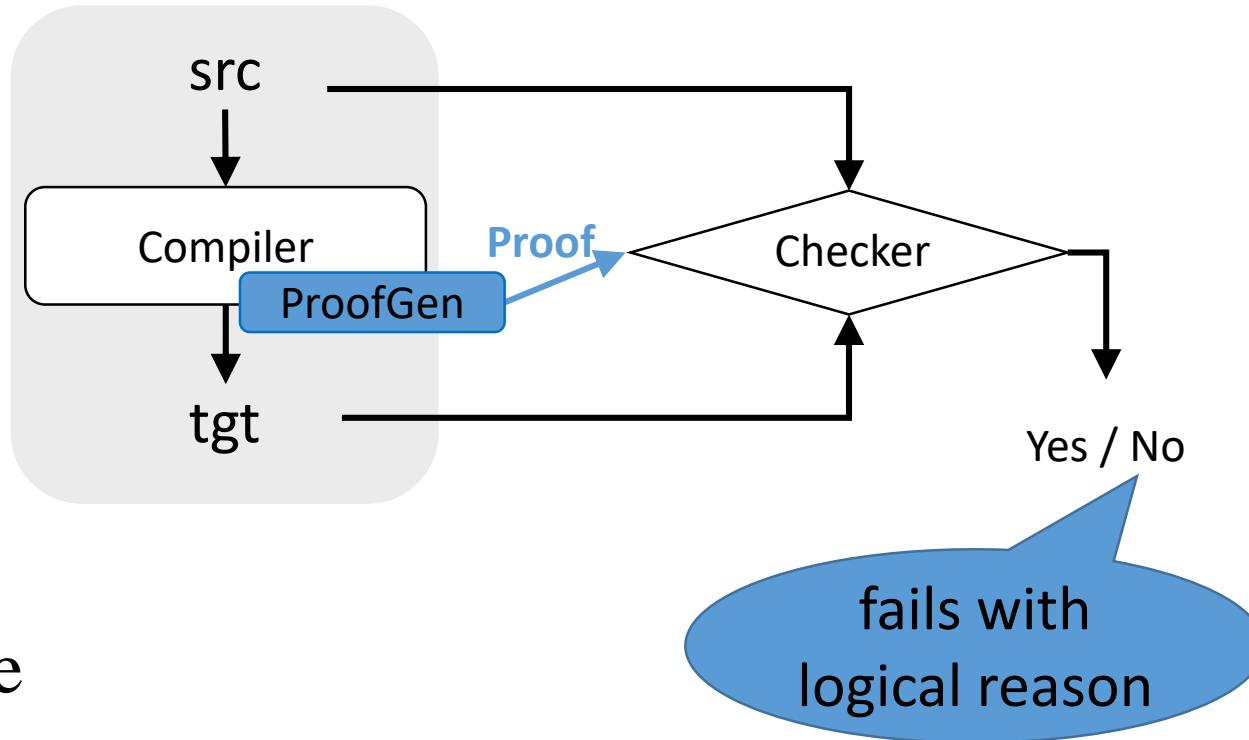
# Approaches to Improving Reliability

- (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]



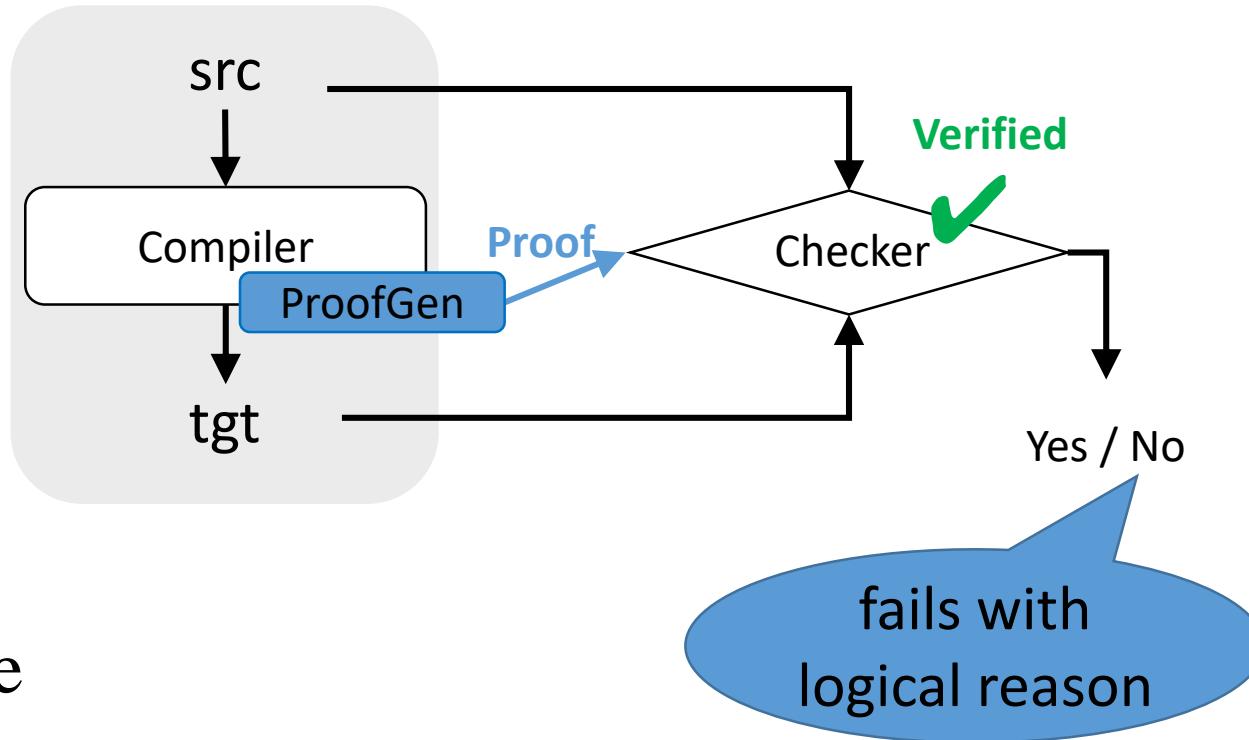
# Approaches to Improving Reliability

- (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(undefined)  
}
```

# Example: A Bug We Found in mem2reg

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

The diagram illustrates a bug found in the `mem2reg` compiler. It shows two versions of a loop. On the left, a red arrow points to the initial allocation of memory for `p`. Inside the loop, `r` is assigned the value at `*p`, `foo(r)` is called, and the value at `*p` is updated to 42. On the right, a red arrow points to the same loop structure, but the assignment `*p := 42` is missing, resulting in undefined behavior for `foo`.

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

⇒

```
loop {
    foo(undefined)
}
```

# 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(undefined)  
}
```

# Example: A Bug We Found in mem2reg

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

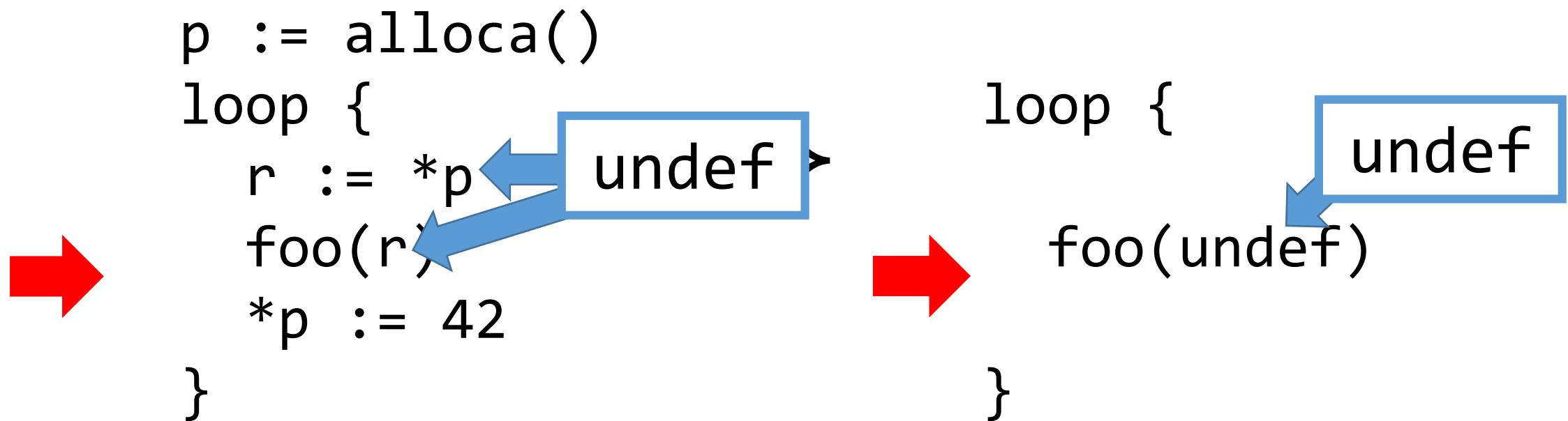
```
p := alloca()  
loop {  
    r := *p  
    foo(r)  
    *p := 42  
}  
  
loop {  
    foo(undefined)  
}
```



A diagram illustrating a bug in memory allocation. On the left, a variable `r` is assigned the value of `*p`. On the right, the value of `*p` is modified to `42`. A blue box labeled `undef` is positioned between the two assignments, with a double-headed arrow indicating it is both undefined and defined, which is a logical inconsistency.

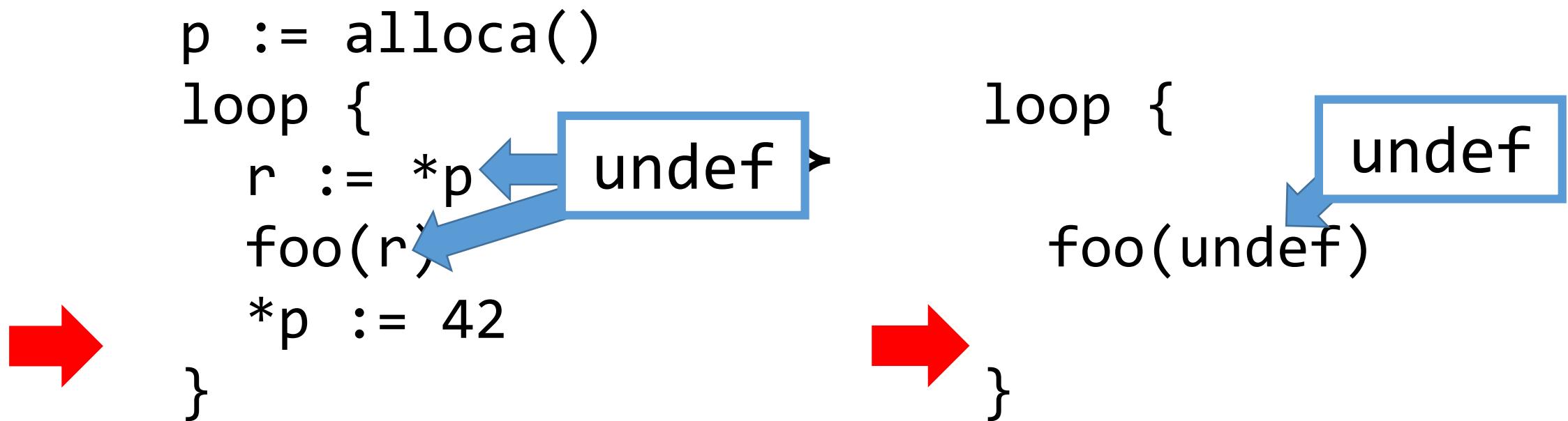
# Example: A Bug We Found in mem2reg

- 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:

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

```

# Example: A Bug We Found in mem2reg

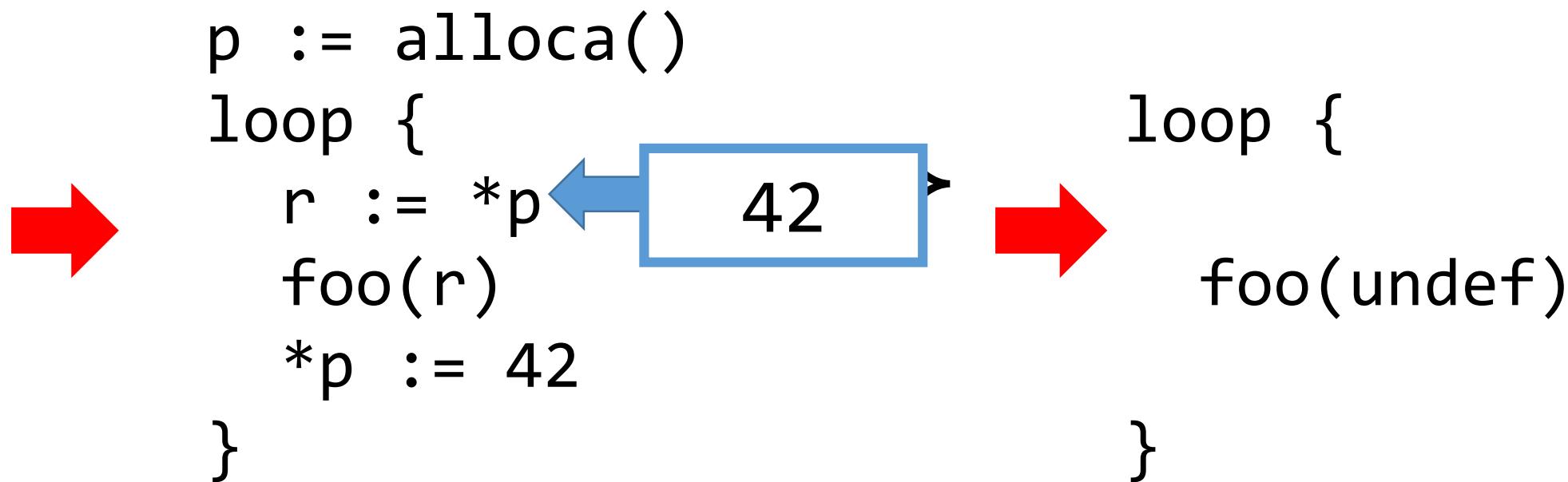
- Credible compilation may detect bugs that testing misses.
- Simplified code from SPEC Benchmark:

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

# Example: A Bug We Found in mem2reg

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

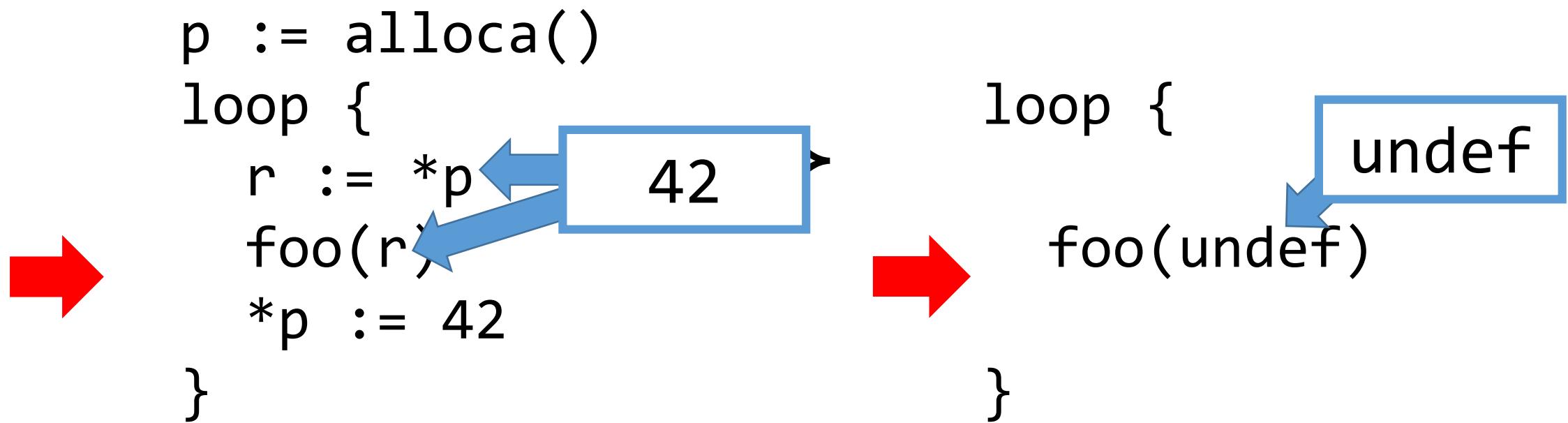
```
p := alloca()
loop {
    r := *p
    foo(r)
    *p := 42
}
loop {
    foo(undefined)
}
```



The diagram illustrates a memory location `p` containing the value `42`. A blue box highlights the value `42`. Two red arrows point to the start of the first loop and the end of the second loop, indicating the flow of control between them.

# Example: A Bug We Found in mem2reg

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



# Example: A Bug in [www.com2reg.com](http://www.com2reg.com)

- Credible
- Simplified

Why testing missed this bug?

Because foo ignores r:

foo(r):

...

s = r & 0x0

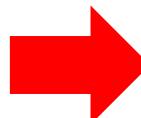
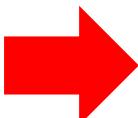
...

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

loop {

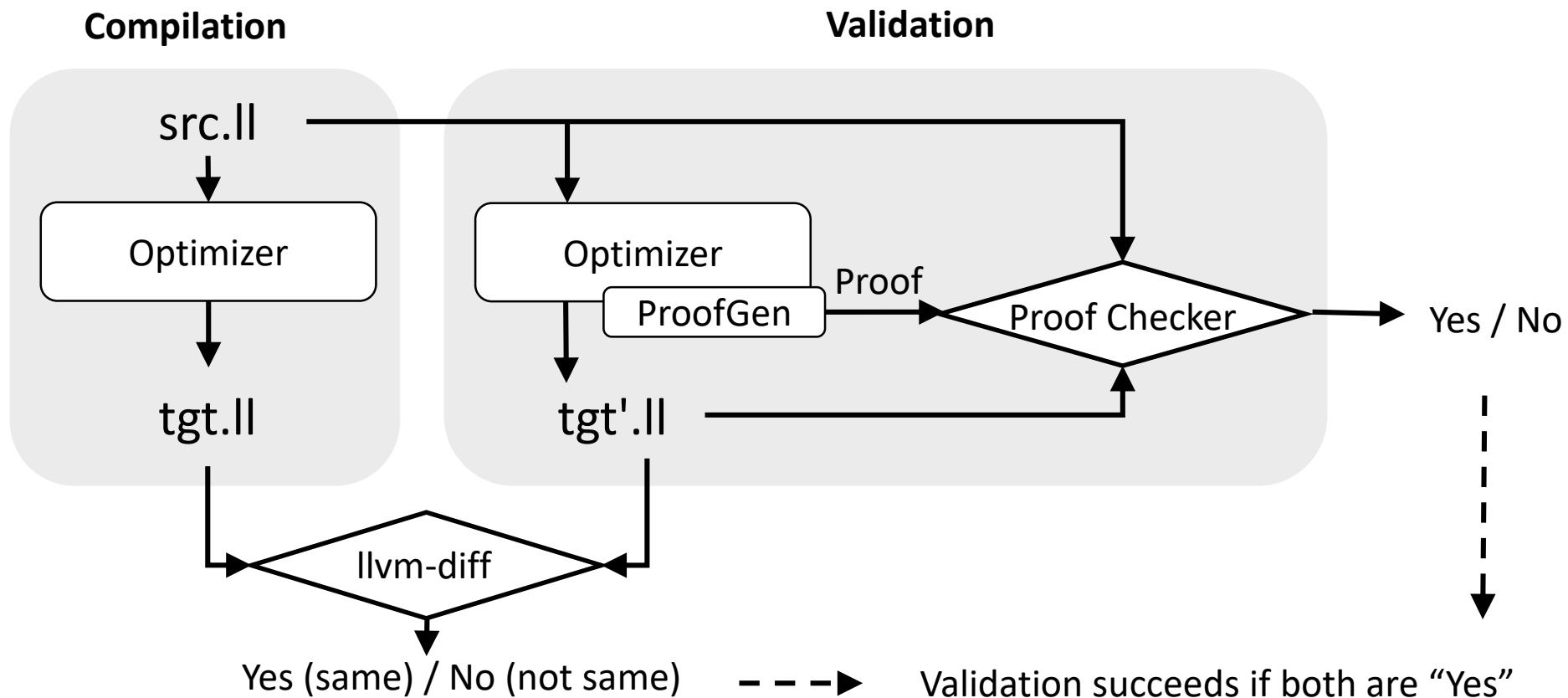
foo(undefined)

undefined

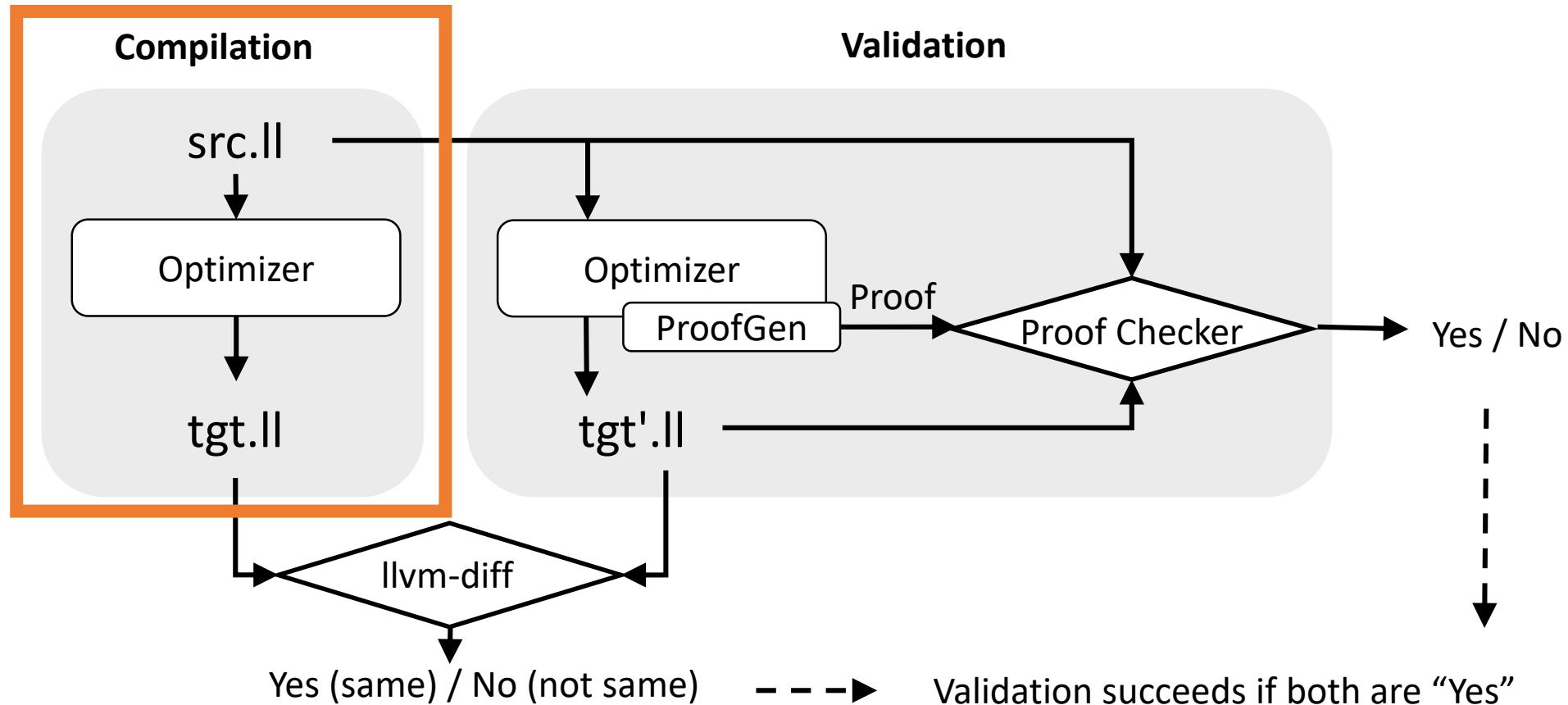


# Crellvm framework

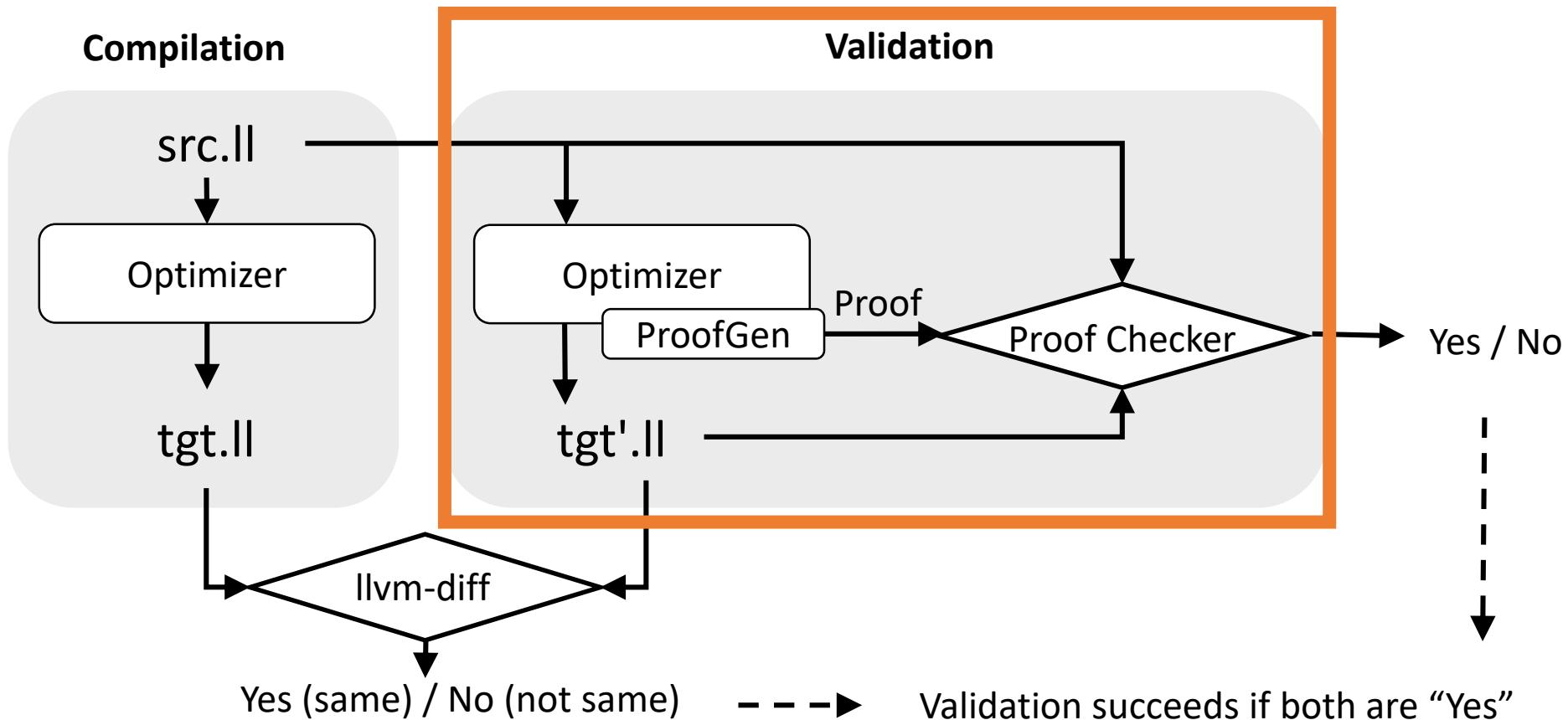
# Crellvm Framework



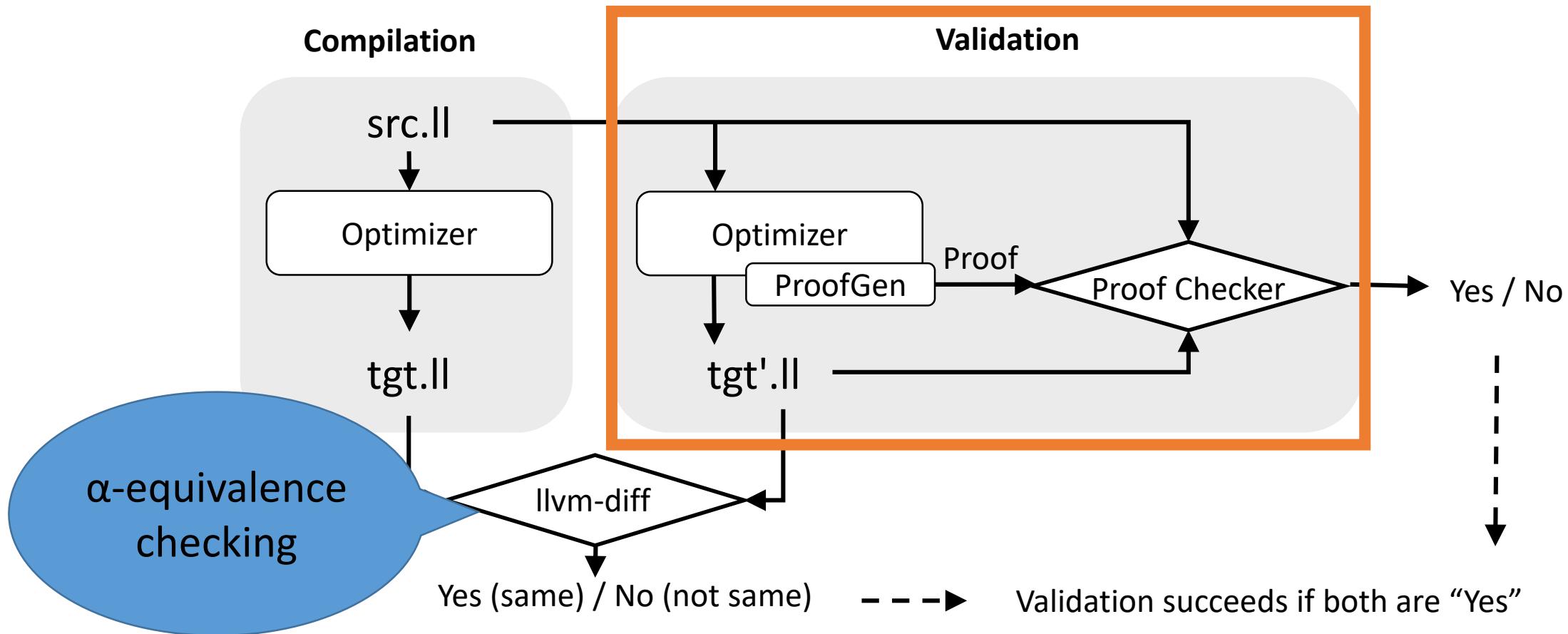
# Crellvm Framework



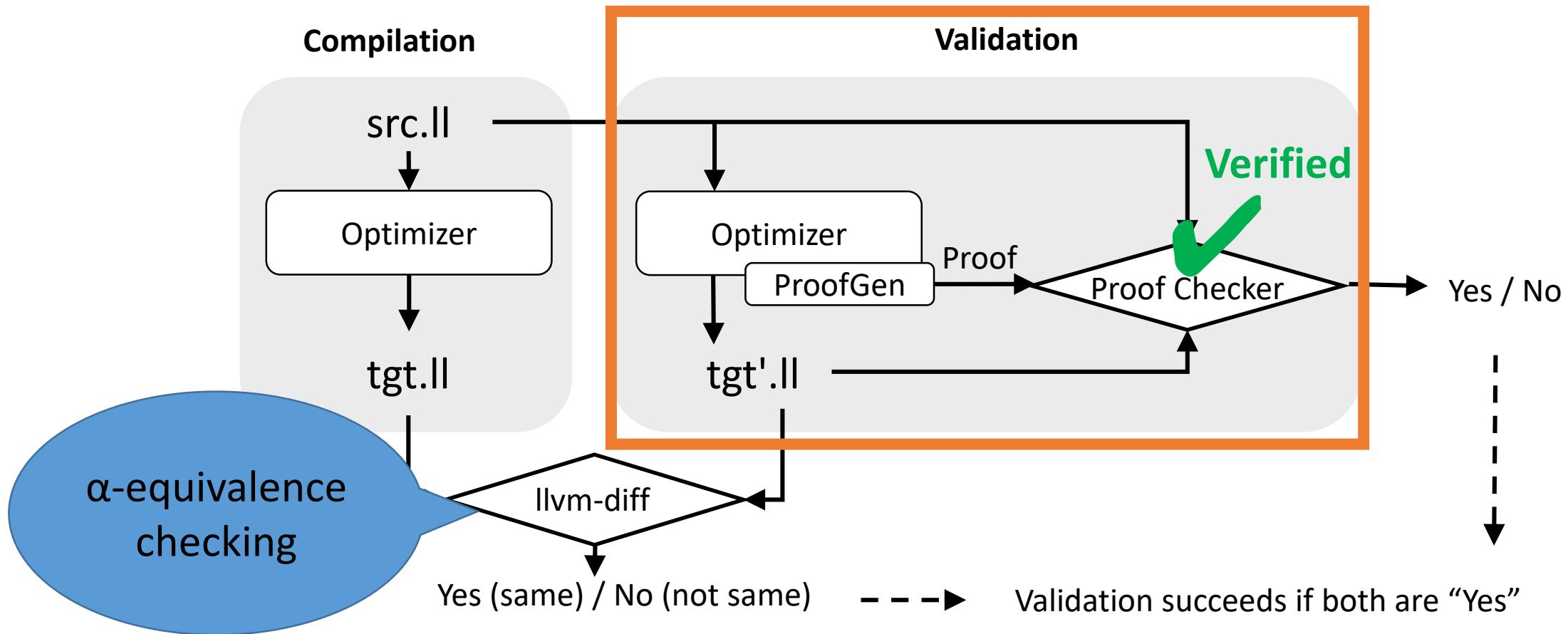
# Crellvm Framework



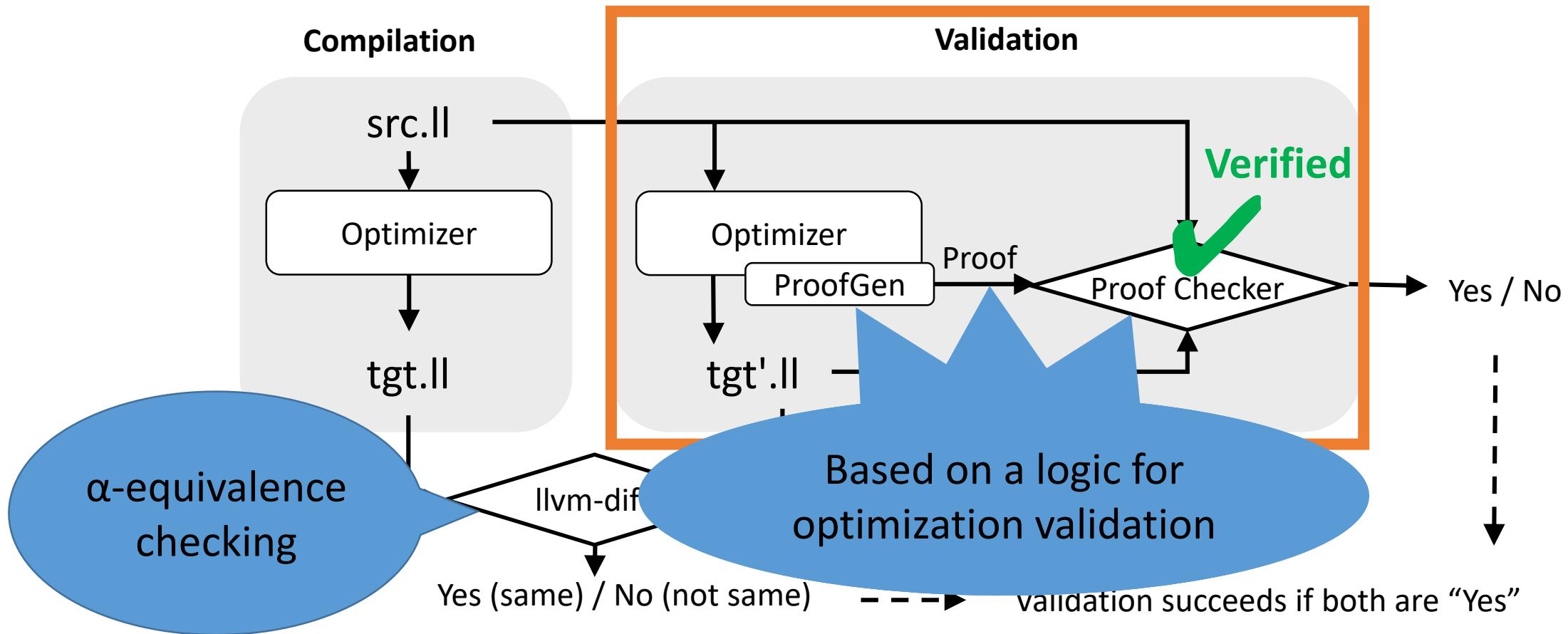
# Crellvm Framework



# Crellvm Framework



# Crellvm Framework



# ERHL: A Logic for Optimization Validation

- Assoc-add optimization in `instcombine`

10 :  $x := \text{add } a \ 1$   $\rightsquigarrow x := \text{add } a \ 1$

$\vdots$   $\rightsquigarrow \vdots$

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

21 :  $\text{foo}(y)$   $\rightsquigarrow \text{foo}(y)$

# ERHL:

Extensible  
Relational  
Hoare  
Logic

## ➤ Assoc-add rule in `instcombine`

$$10 : \boxed{x := \text{add } a \ 1} \rightsquigarrow \boxed{x := \text{add } a \ 1}$$

$$\vdots \rightsquigarrow \vdots$$

$$20 : \boxed{y := \text{add } x \ 2} \rightsquigarrow \boxed{y := \text{add } a \ 3}$$

$$21 : \boxed{\text{foo}(y)} \rightsquigarrow \boxed{\text{foo}(y)}$$

# ERHL: A Logic for Optimization Validation

- Assoc-add optimization in `instcombine`

10 :  $x := \text{add } a \ 1$   $\rightsquigarrow x := \text{add } a \ 1$

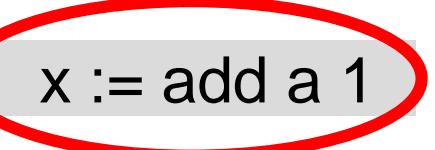
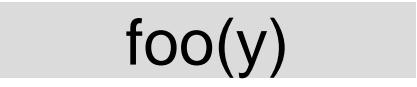
$\vdots$   $\rightsquigarrow \vdots$

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

21 :  $\text{foo}(y)$   $\rightsquigarrow \text{foo}(y)$

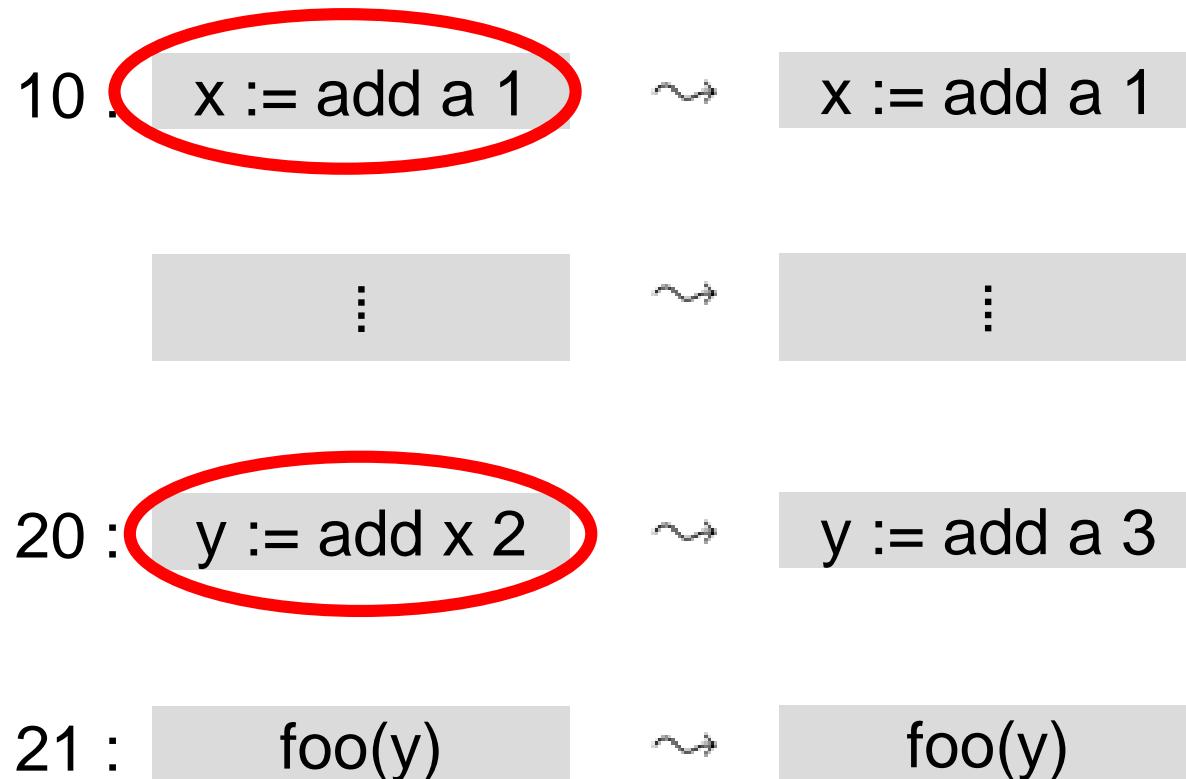
# ERHL: A Logic for Optimization Validation

- Assoc-add optimization in `instcombine`

10 :  <code>x := add a 1</code>	~	<code>x := add a 1</code>
 :	~	 :
20 :  <code>y := add x 2</code>	~	<code>y := add a 3</code>
21 :  <code>foo(y)</code>	~	<code>foo(y)</code>

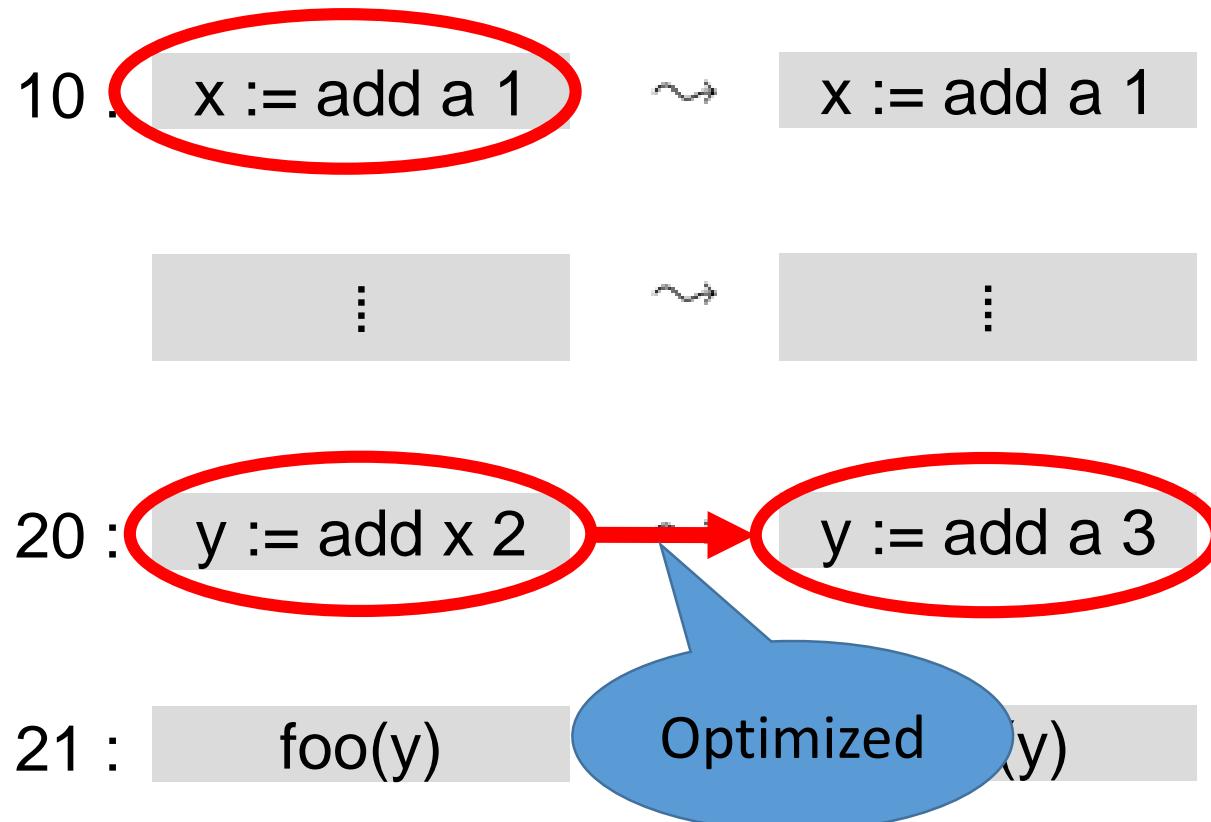
# ERHL: A Logic for Optimization Validation

- Assoc-add optimization in `instcombine`



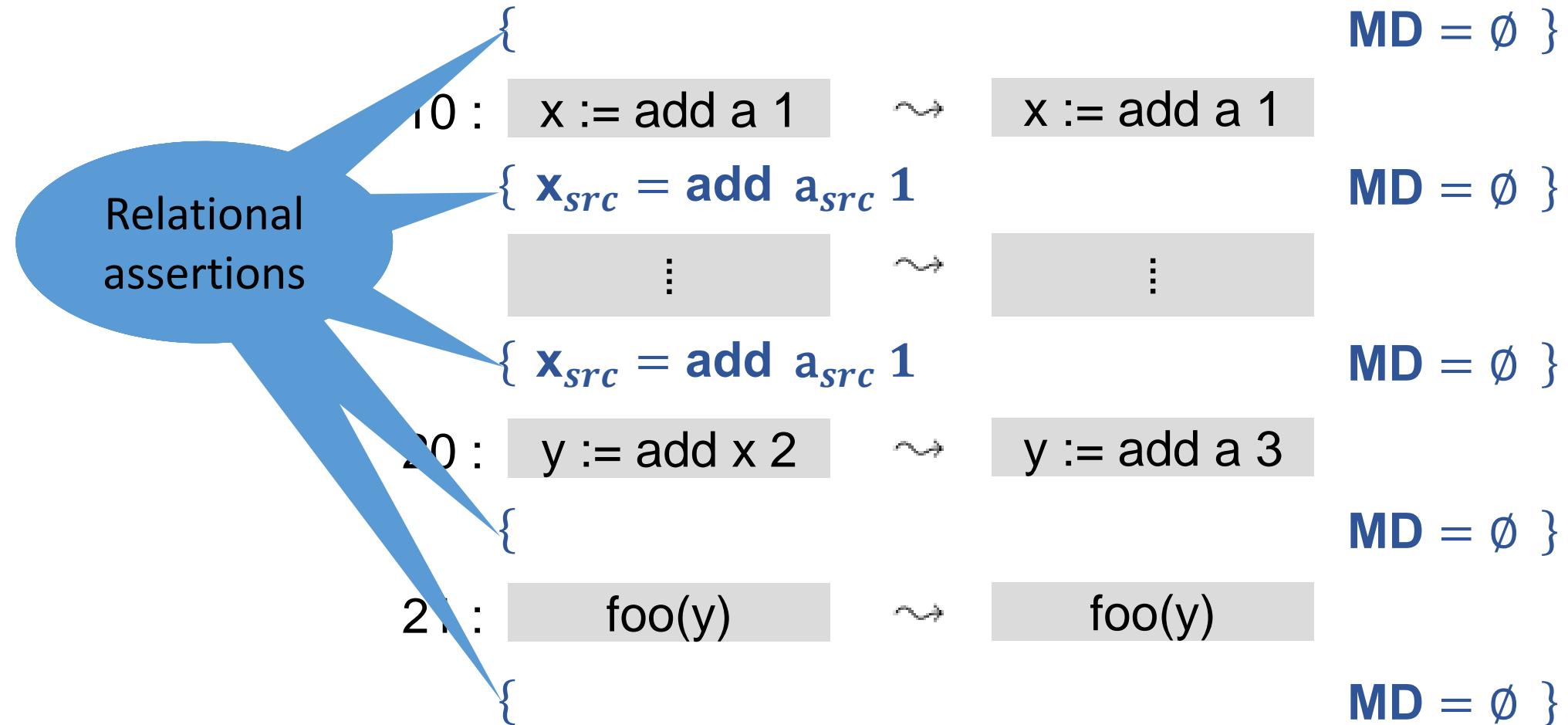
# ERHL: A Logic for Optimization Validation

- Assoc-add optimization in `instcombine`



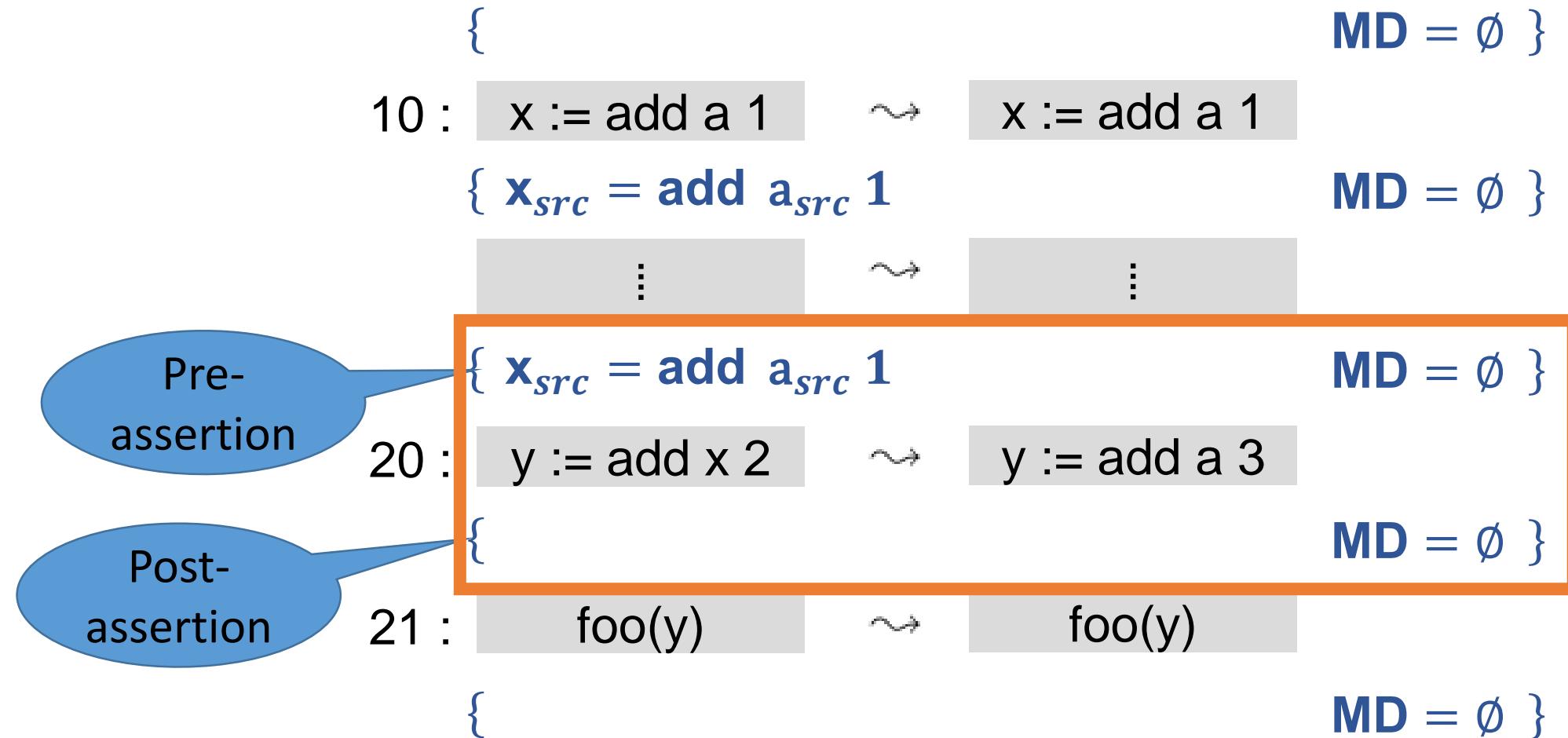
# ERHL: A Logic for Optimization Validation

- Assoc-add optimization in `instcombine`



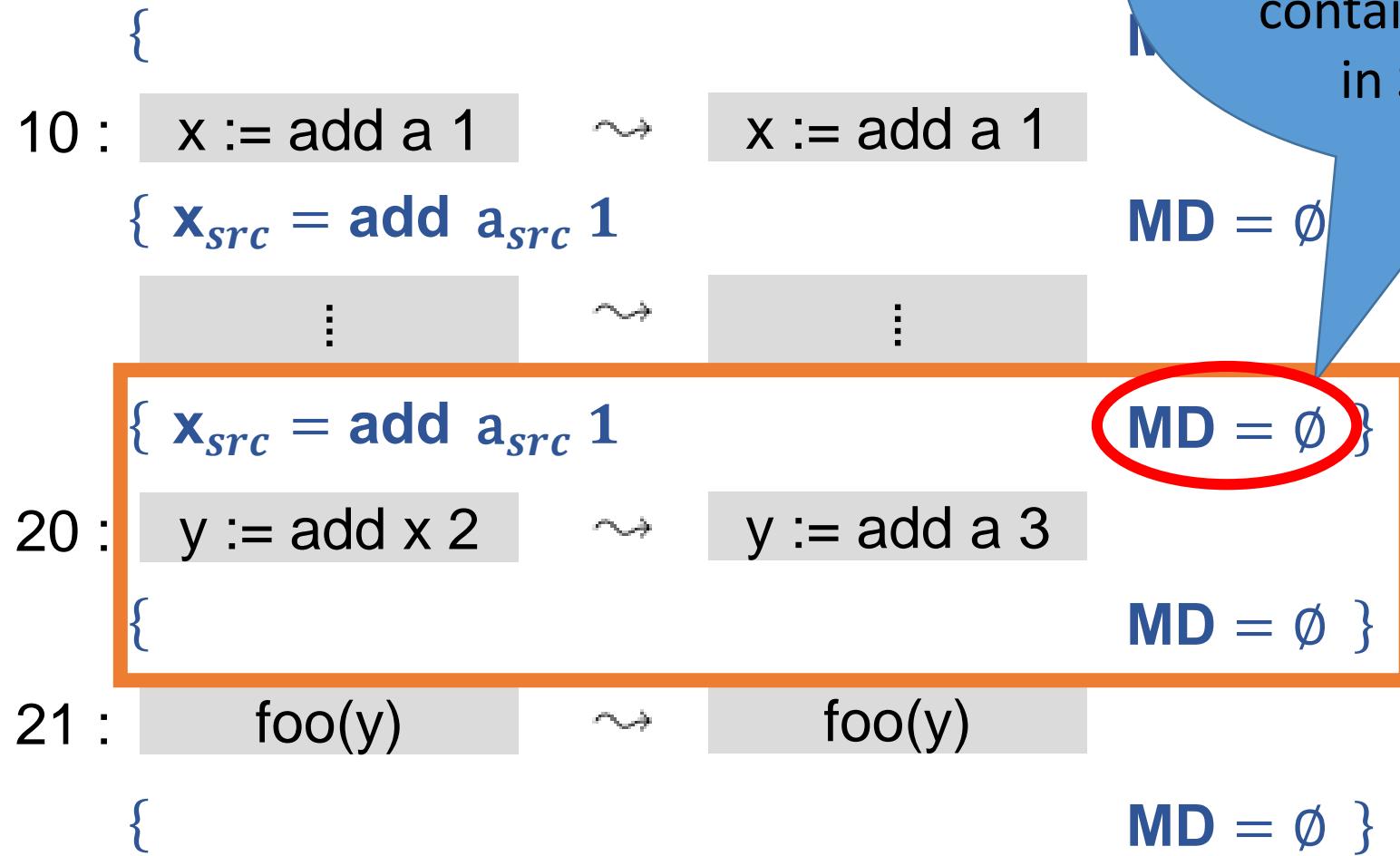
# ERHL: A Logic for Optimization Validation

- Assoc-add optimization in `instcombine`



# ERHL: A Logic for Optimization Validation

- Assoc-add optimization in `instcombine`



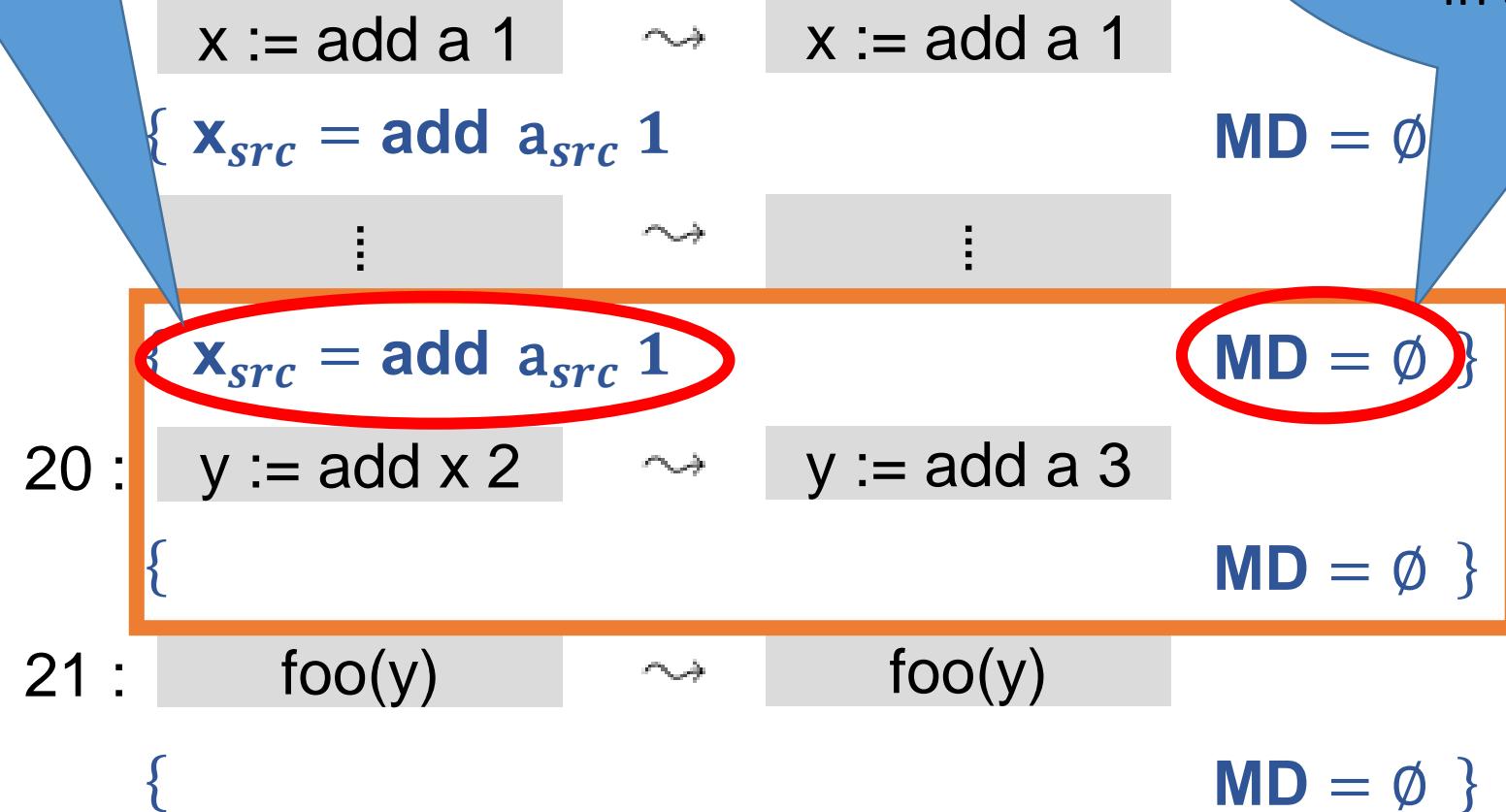
# EPIII : A Logic for Optimization Validation

(Unary Property)

This equation holds  
in SRC

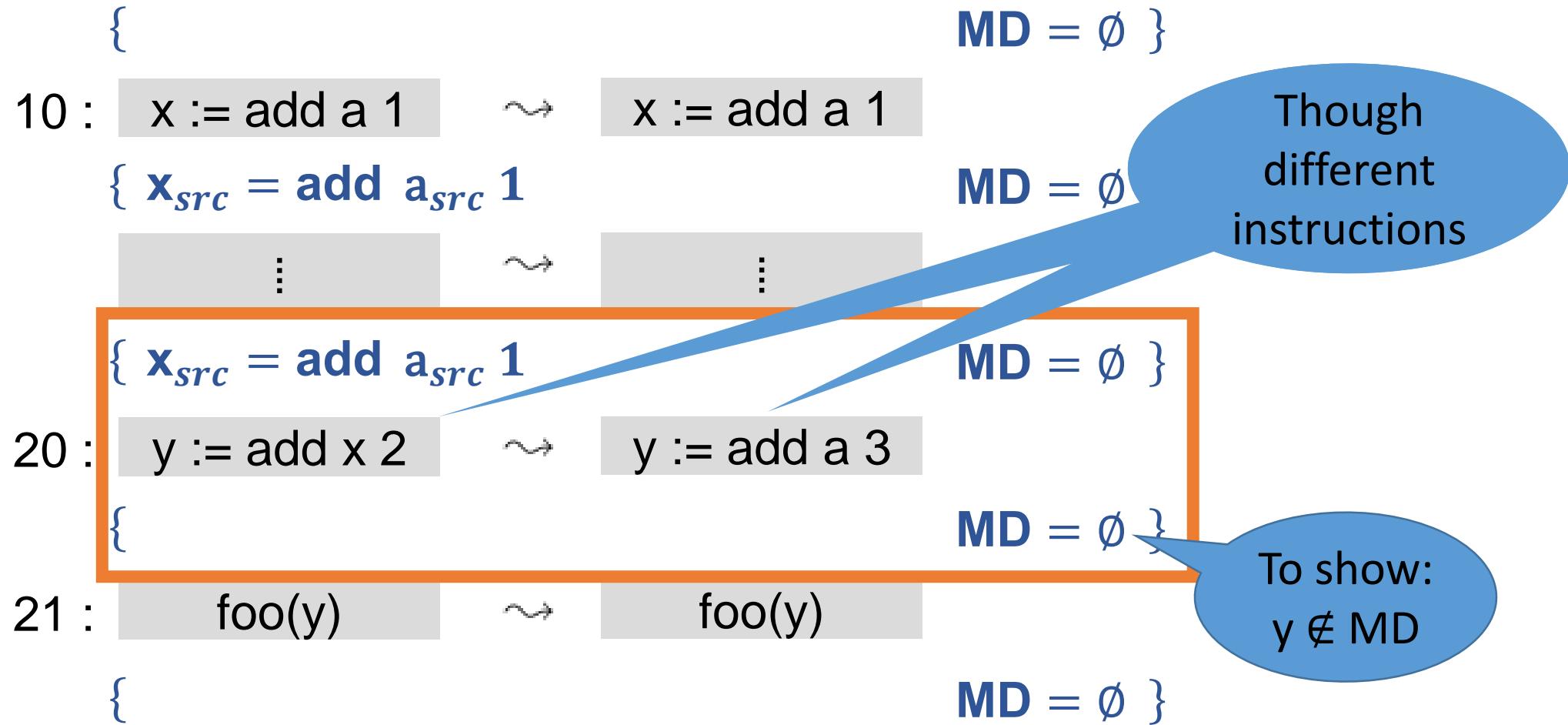
(Relational Property)

All registers  
contain same value  
in SRC & TGT



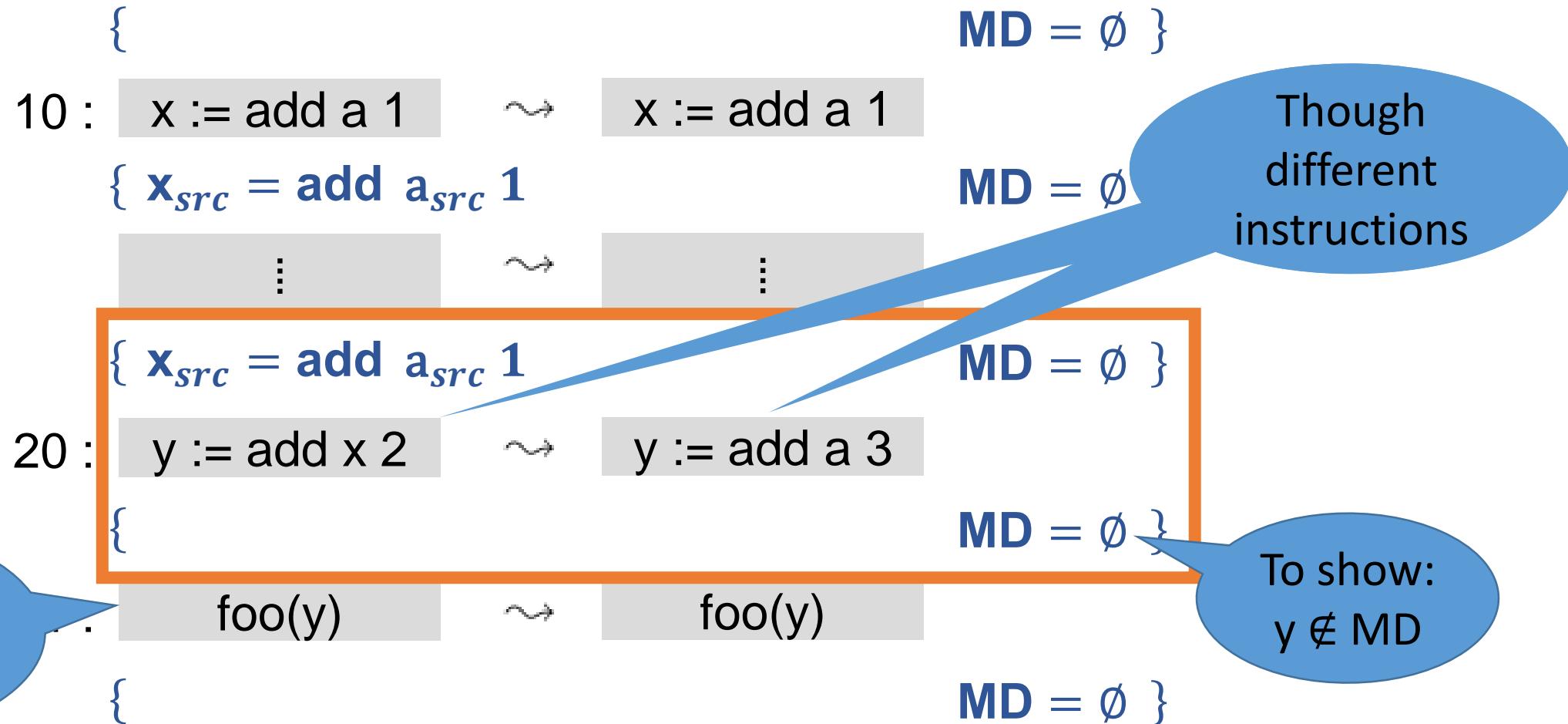
# ERHL: A Logic for Optimization Validation

- 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

{  
  10 : x := add a 1    =  $\emptyset$  }  
  { x = add a 1  
    :  
    :  
  }

10 : x := add a 1    =  $\emptyset$  }  
  { x = add a 1  
    :  
    :  
  }

{  $x_{src} = \text{add } a_{src} 1$     **MD =  $\emptyset$**  }  
  20 : y := add x 2    ~  
  y := add a 3

{  
  21 : foo(y)    ~  
  foo(y)  
  **MD =  $\emptyset$**  }

{  
  21 : foo(y)    ~  
  foo(y)  
  **MD =  $\emptyset$**  }

# ERHL: A Logic for Optimization Validation

- Assoc-add op

(Design Choice)

MD is the only relational property

{  
10 : x = add a 1      MD =  $\emptyset$  }

{ x = add a 1      MD =  $\emptyset$  }  
• Proof checking is simple

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

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

{      MD =  $\emptyset$  }

21 : foo(y)       $\rightsquigarrow$       foo(y)

{      MD =  $\emptyset$  }

# ERHL: A Logic for Optimization Validation

- Assoc-add op

(Design Choice)

MD is the only relational property

{  
10 : x = add a 1  
{ x = add a 1  
=  $\emptyset$  }

• Proof checking is simple  
• Still expressive enough  
{ x = add a 1  
=  $\emptyset$  }

{ x<sub>src</sub> = add a<sub>src</sub> 1    MD =  $\emptyset$  }

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

{    MD =  $\emptyset$  }

21 : foo(y)       $\rightsquigarrow$       foo(y)

{    MD =  $\emptyset$  }

# ERHL: A Logic for Optimization Validation

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

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

{ } MD =  $\emptyset$

# ERHL: A Logic for Optimization Validation

Pre-  
assertion

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

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

Post-  
assertion

$MD = \emptyset$  }

# ERHL: A Logic for Optimization Validation

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

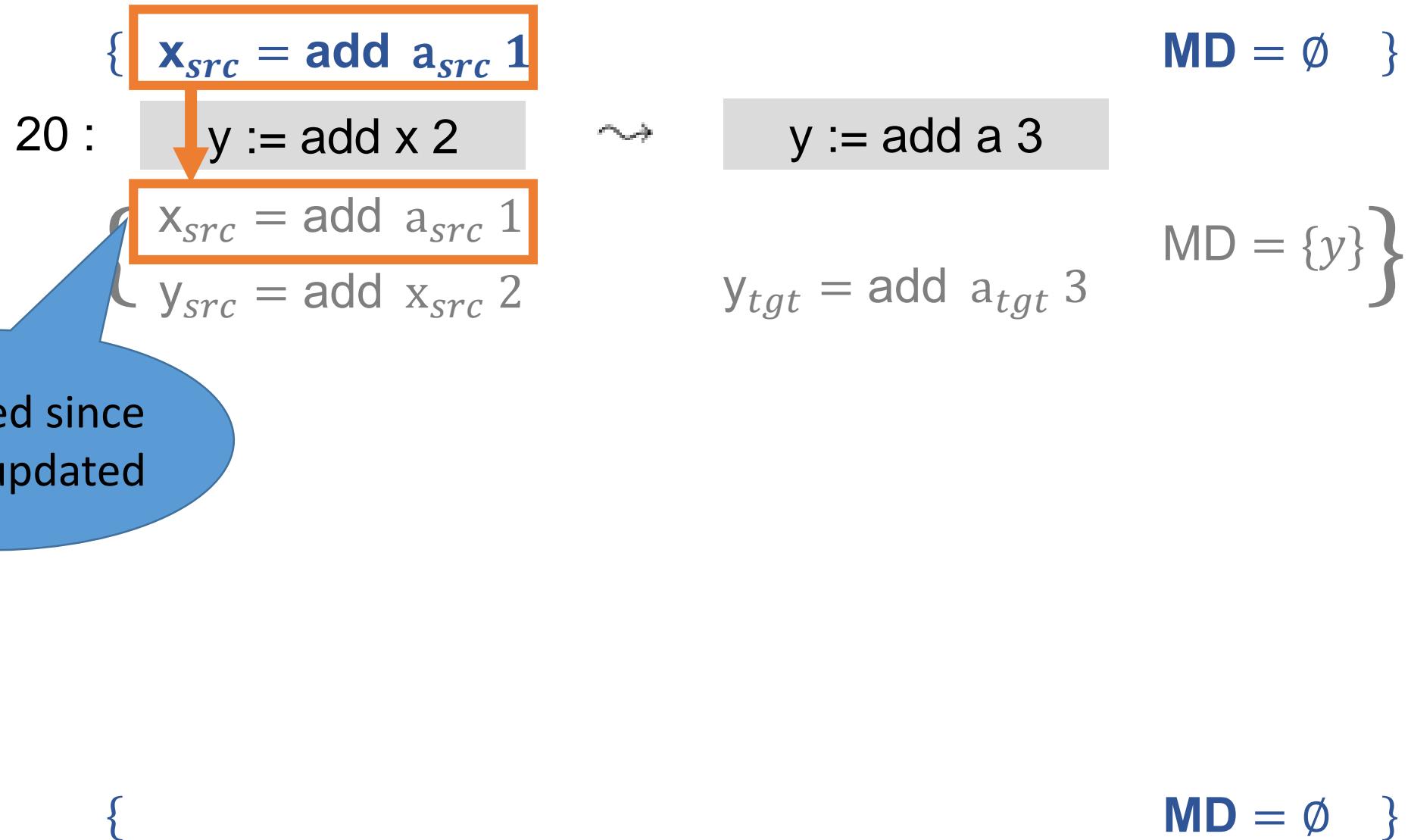
20 :  $y := \mathbf{add} \ x \ 2 \rightsquigarrow y := \mathbf{add} \ a \ 3$

Strong post-condition

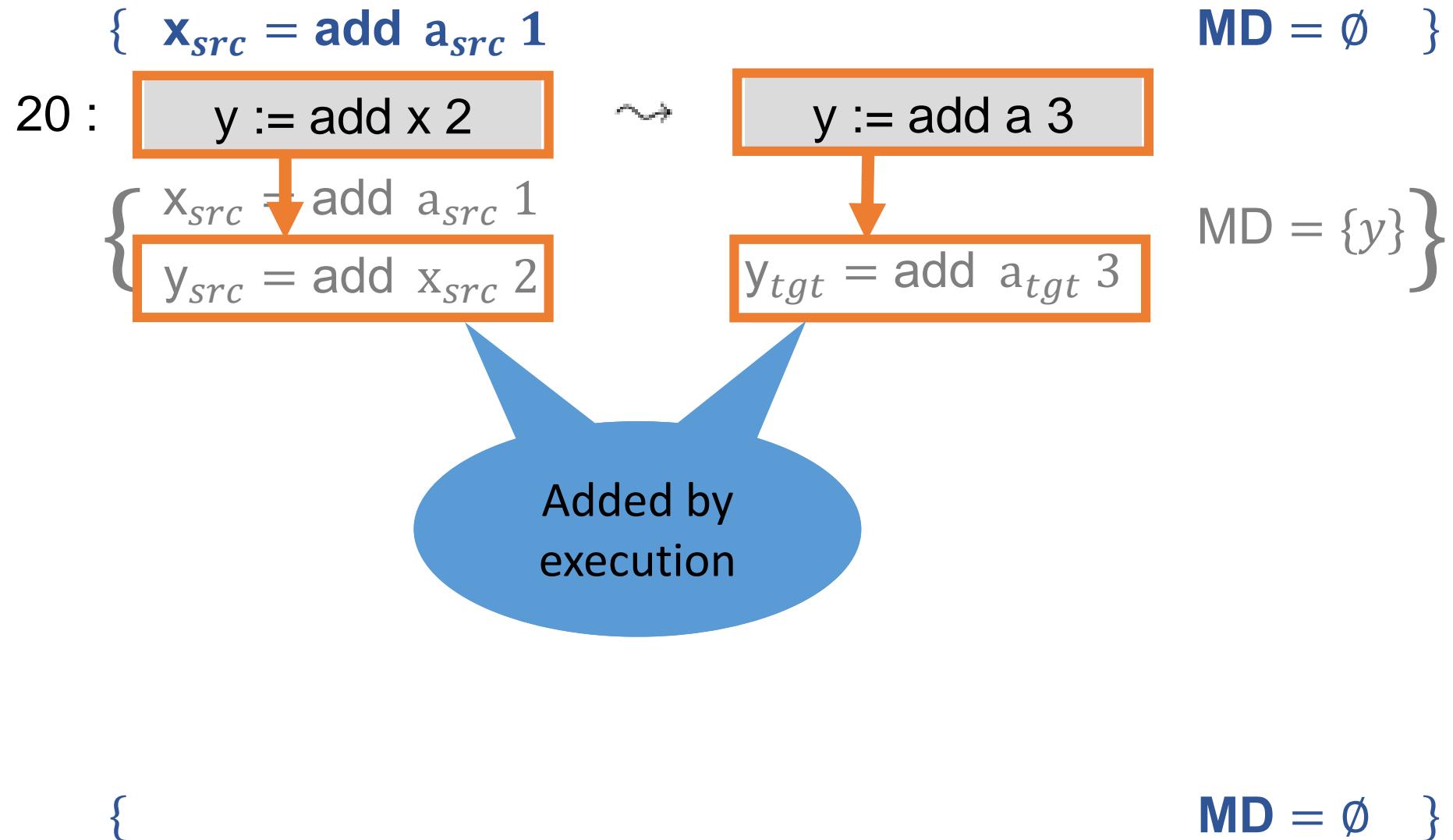
$$\left\{ \begin{array}{l} \mathbf{x}_{src} = \mathbf{add} \ a_{src} \ 1 \\ \mathbf{y}_{src} = \mathbf{add} \ x_{src} \ 2 \end{array} \right. \quad \mathbf{y}_{tgt} = \mathbf{add} \ a_{tgt} \ 3 \quad \mathbf{MD} = \{y\}$$

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

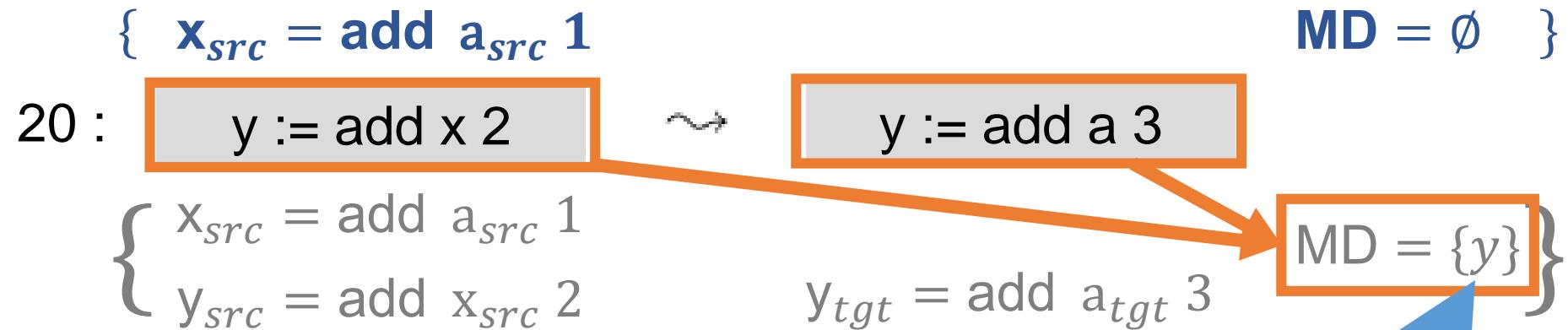
# ERHL: A Logic for Optimization Validation



# ERHL: A Logic for Optimization Validation



# ERHL: A Logic for Optimization Validation



y added to MD since  
y updated differently

{

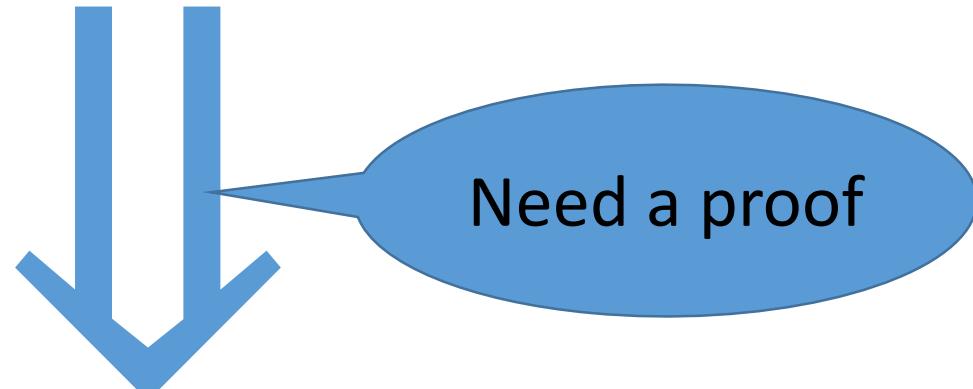
 $\text{MD} = \emptyset \quad \}$

# ERHL: A Logic for Optimization Validation

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

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

{  $x_{src} = \text{add } a_{src} 1$   
 $y_{src} = \text{add } x_{src} 2$        $y_{tgt} = \text{add } a_{tgt} 3$        $\text{MD} = \{y\}$  }



{       $\text{MD} = \emptyset$  }

# ERHL: A Logic for Optimization Validation

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

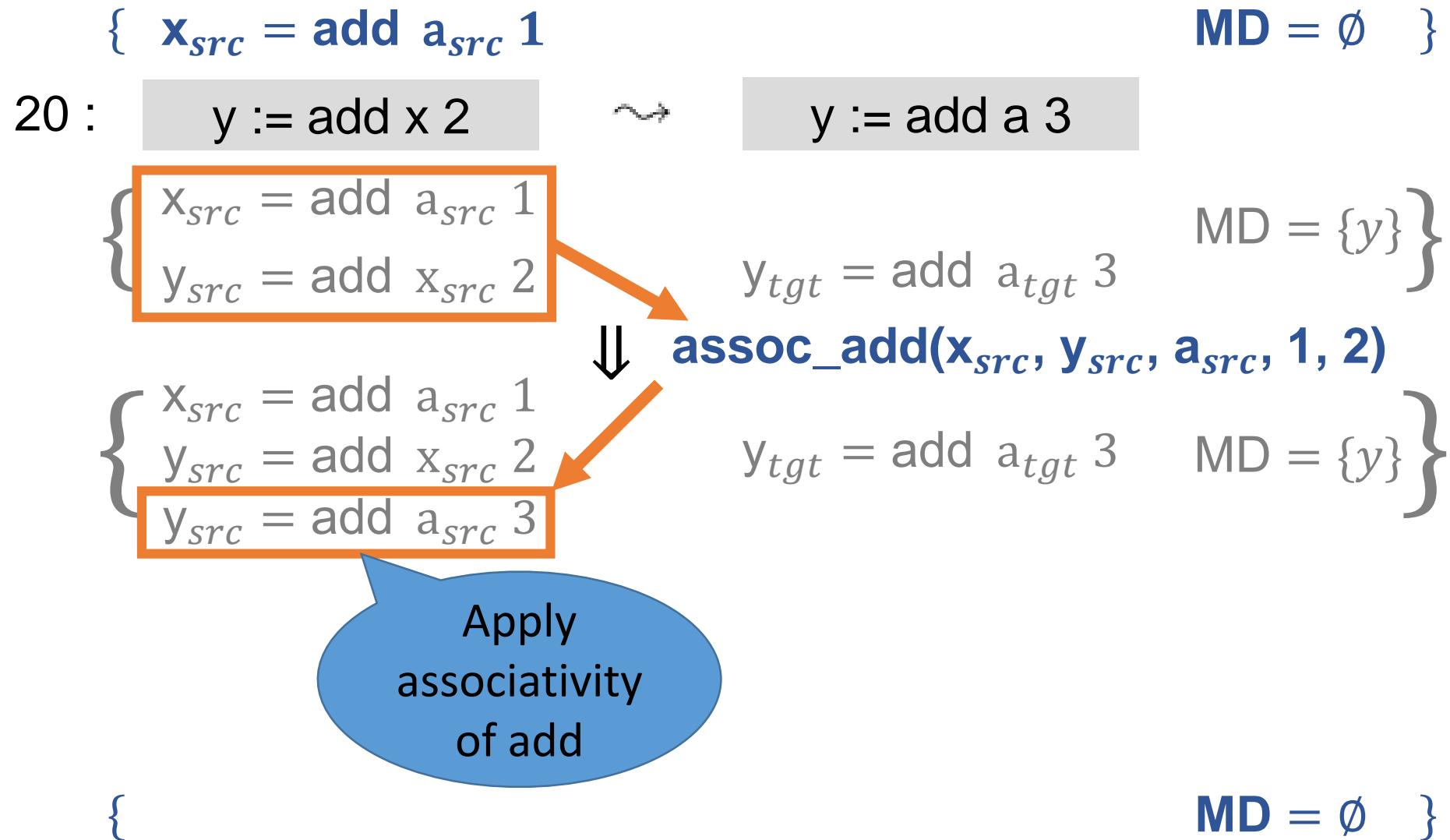
20 :  $y := \text{add } x \; 2$   $\rightsquigarrow y := \text{add } a \; 3$

$$\left\{ \begin{array}{l} \mathbf{x}_{src} = \mathbf{add} \; \mathbf{a}_{src} \; 1 \\ \mathbf{y}_{src} = \mathbf{add} \; \mathbf{x}_{src} \; 2 \end{array} \right. \quad y_{tgt} = \text{add } a_{tgt} \; 3 \quad \mathbf{MD} = \{y\} \Big\}$$

$$\downarrow \quad \mathbf{assoc\_add(x}_{src}, \mathbf{y}_{src}, \mathbf{a}_{src}, \mathbf{1}, \mathbf{2})$$
$$\left. \begin{array}{l} y_{tgt} = \text{add } a_{tgt} \; 3 \\ \mathbf{MD} = \{y\} \end{array} \right\}$$
$$\left\{ \begin{array}{l} \mathbf{x}_{src} = \mathbf{add} \; \mathbf{a}_{src} \; 1 \\ \mathbf{y}_{src} = \mathbf{add} \; \mathbf{x}_{src} \; 2 \\ \mathbf{y}_{src} = \mathbf{add} \; \mathbf{a}_{src} \; 3 \end{array} \right.$$

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

# ERHL: A Logic for Optimization Validation



# ERHL: A Logic for Optimization Validation

$$20 : \quad \left\{ \begin{array}{l} \mathbf{x}_{src} = \mathbf{add} \ a_{src} 1 \\ y := \mathbf{add} \ x 2 \end{array} \right. \rightsquigarrow \left. \begin{array}{l} y := \mathbf{add} \ a 3 \end{array} \right. \quad \mathbf{MD} = \emptyset$$

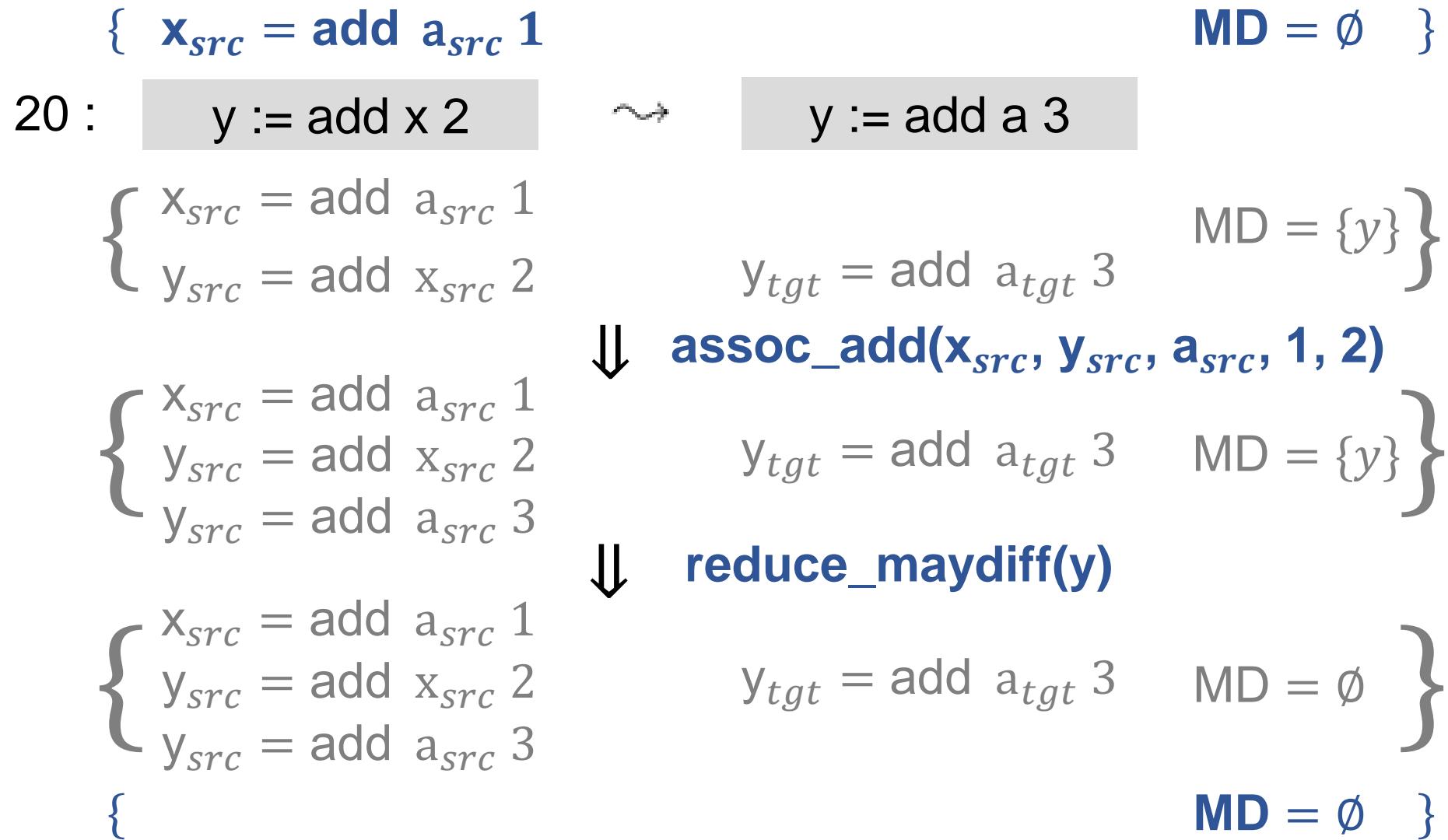
$$\left\{ \begin{array}{l} \mathbf{x}_{src} = \mathbf{add} \ a_{src} 1 \\ y_{src} = \mathbf{add} \ x_{src} 2 \end{array} \right. \quad \left. \begin{array}{l} y_{tgt} = \mathbf{add} \ a_{tgt} 3 \\ \mathbf{MD} = \{y\} \end{array} \right\}$$

$$\left\{ \begin{array}{l} \mathbf{x}_{src} = \mathbf{add} \ a_{src} 1 \\ y_{src} = \mathbf{add} \ x_{src} 2 \\ y_{src} = \mathbf{add} \ a_{src} 3 \end{array} \right. \quad \downarrow \quad \left. \begin{array}{l} \mathbf{assoc\_add(x}_{src}, y_{src}, a_{src}, 1, 2) \\ y_{tgt} = \mathbf{add} \ a_{tgt} 3 \\ \mathbf{MD} = \{y\} \end{array} \right\}$$

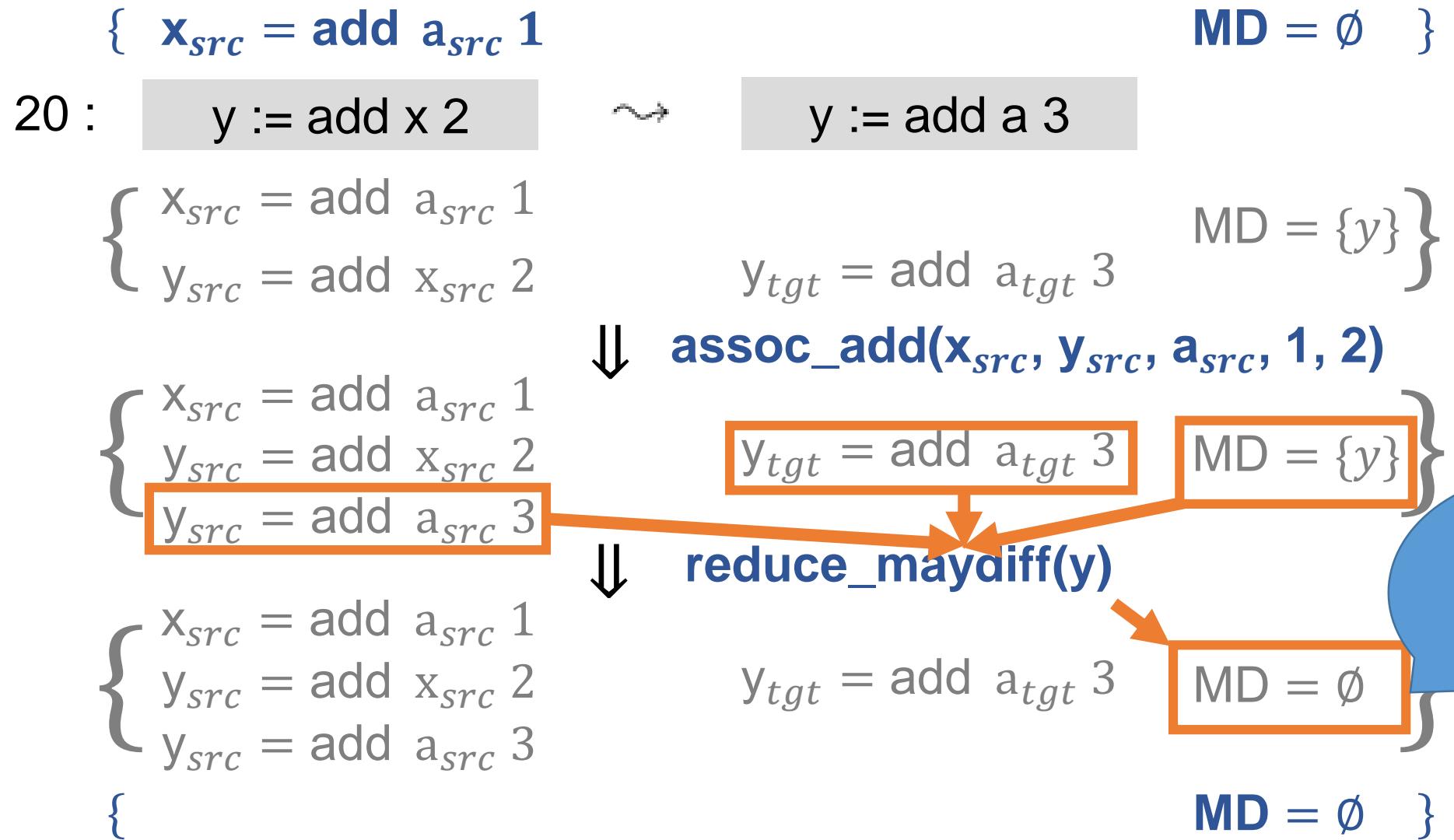
Propagated

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

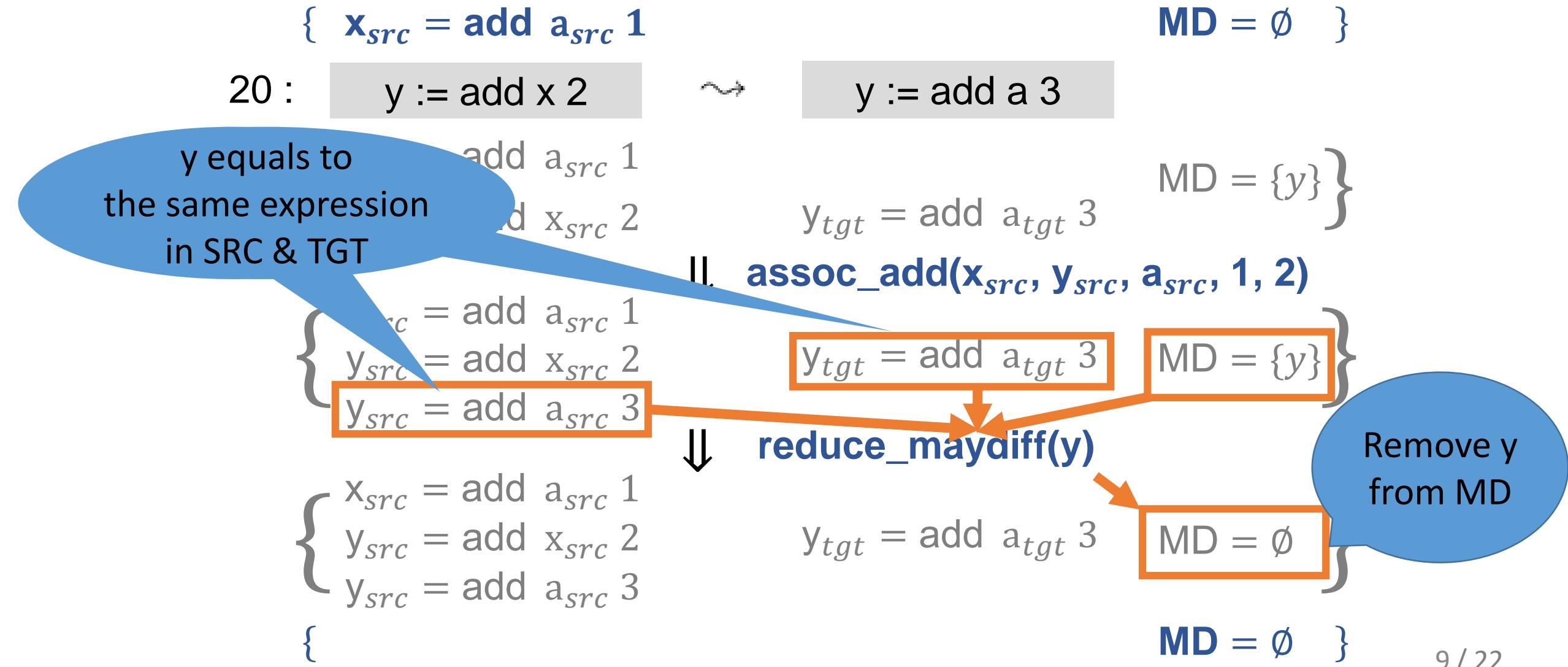
# ERHL: A Logic for Optimization Validation



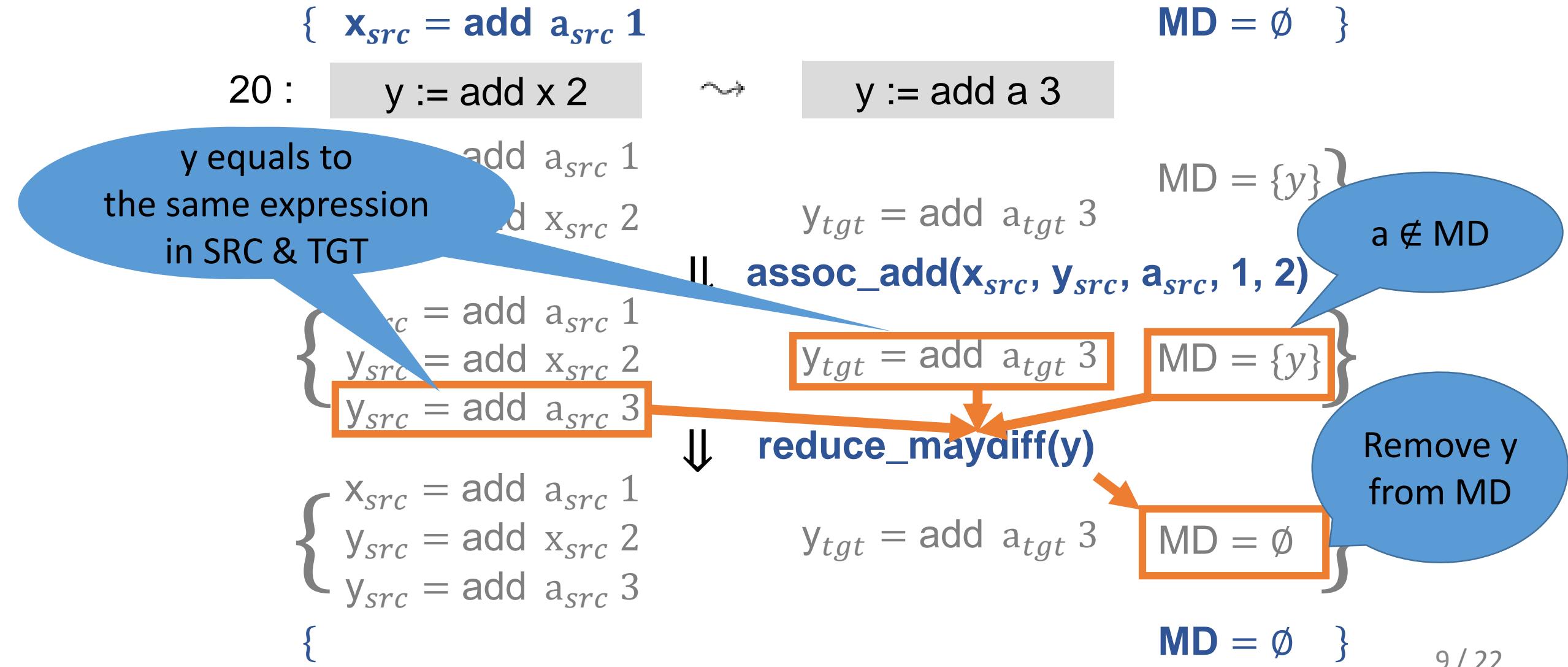
# ERHL: A Logic for Optimization Validation



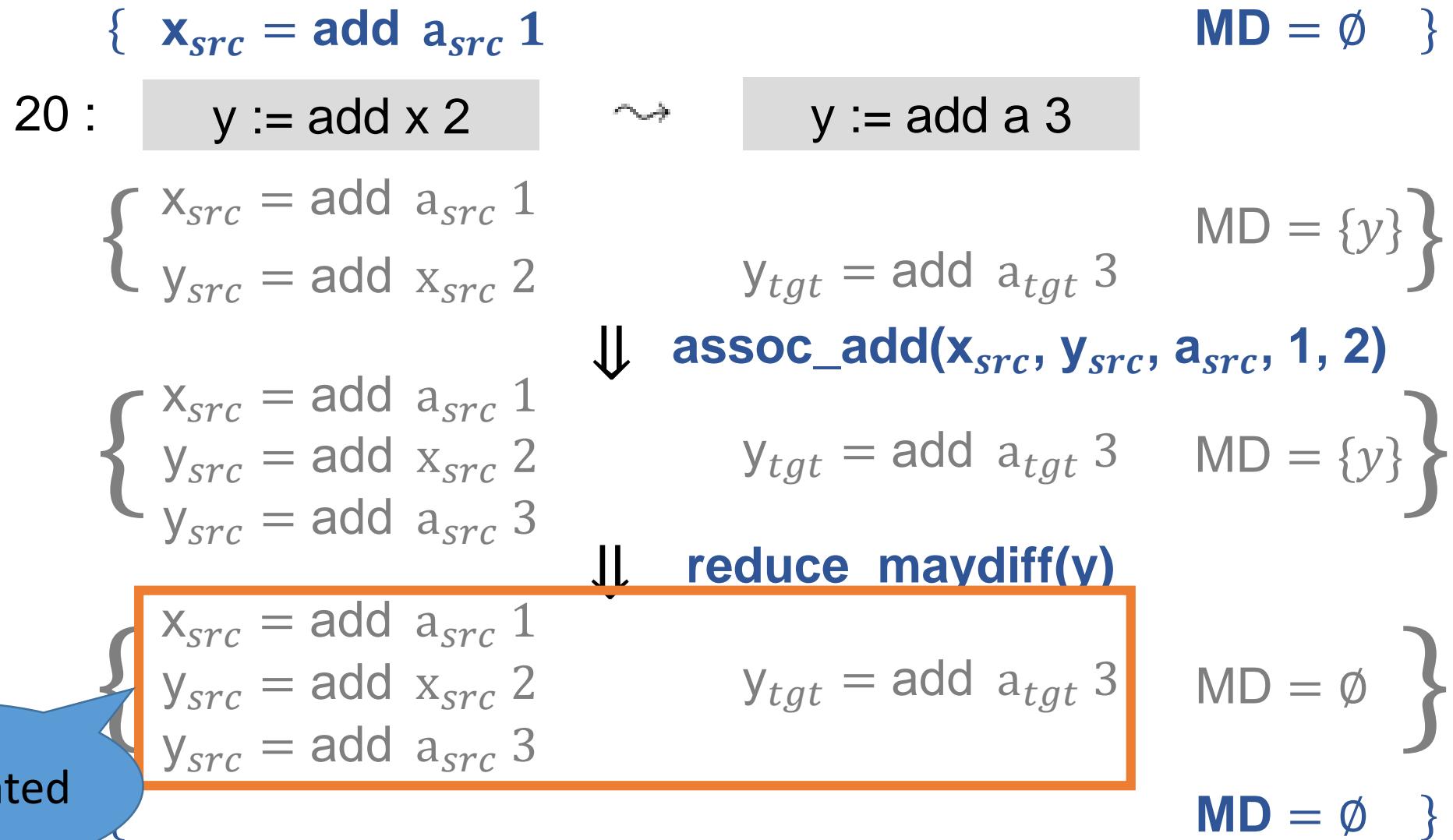
# ERHL: A Logic for Optimization Validation



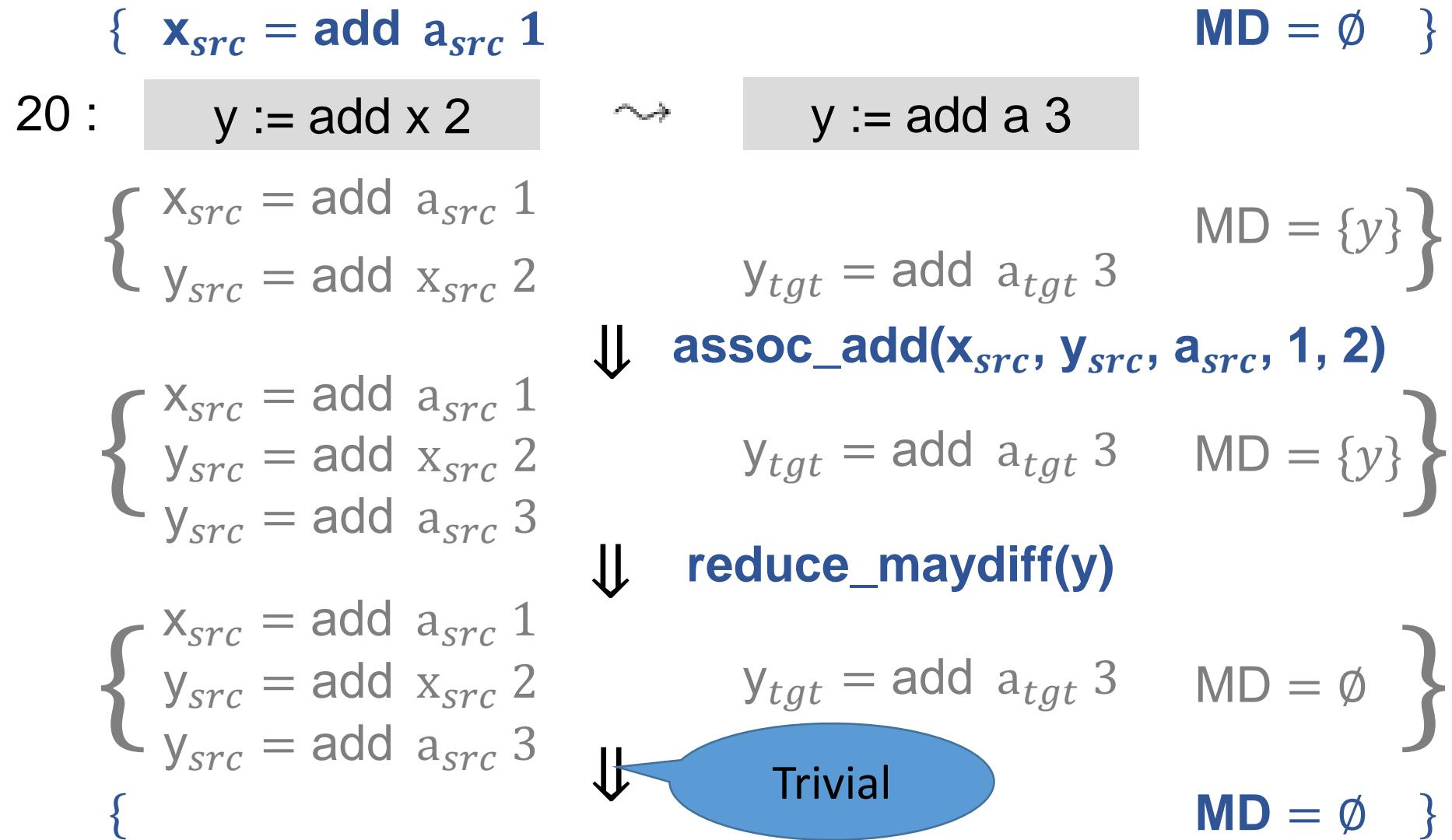
# ERHL: A Logic for Optimization Validation



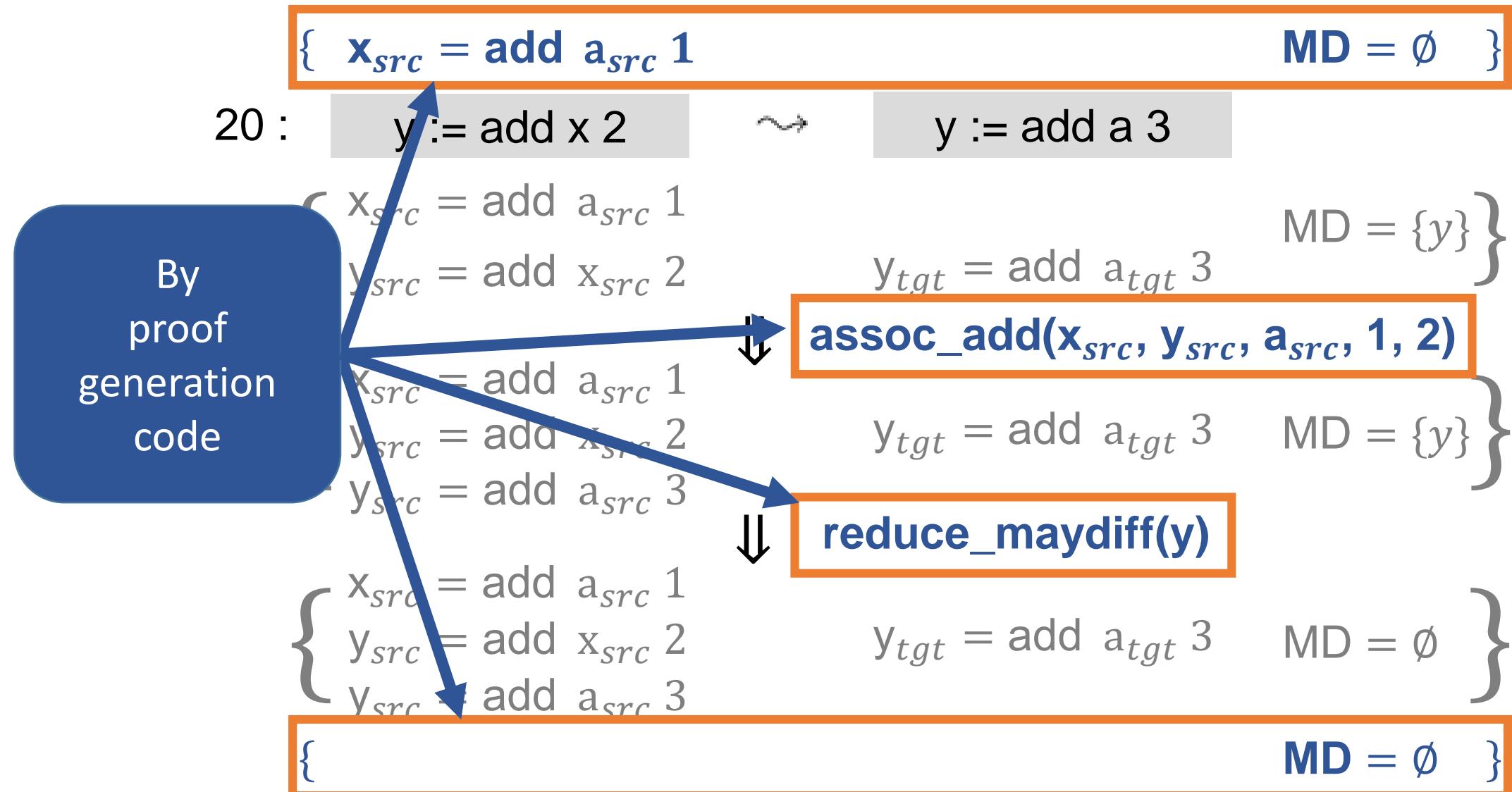
# ERHL: A Logic for Optimization Validation



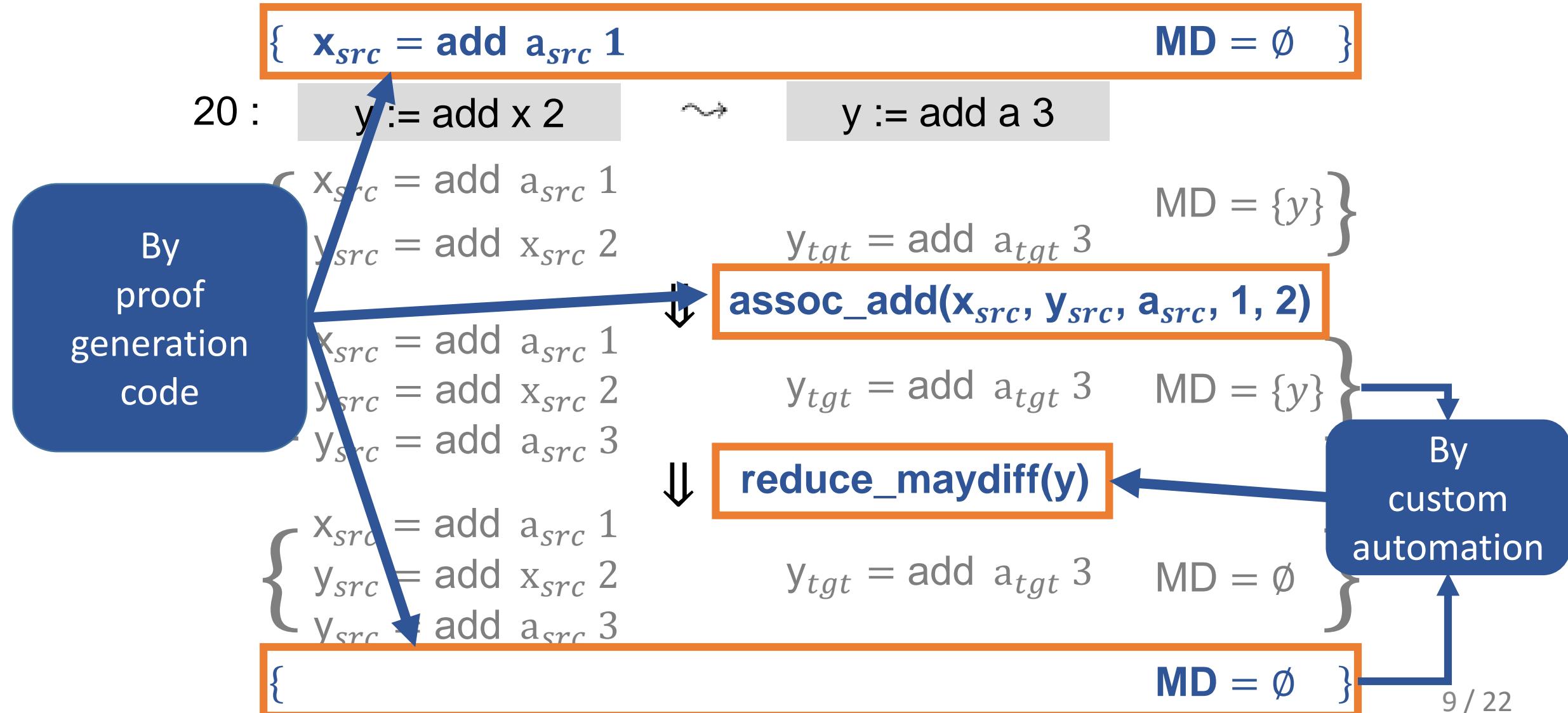
# ERHL: A Logic for Optimization Validation



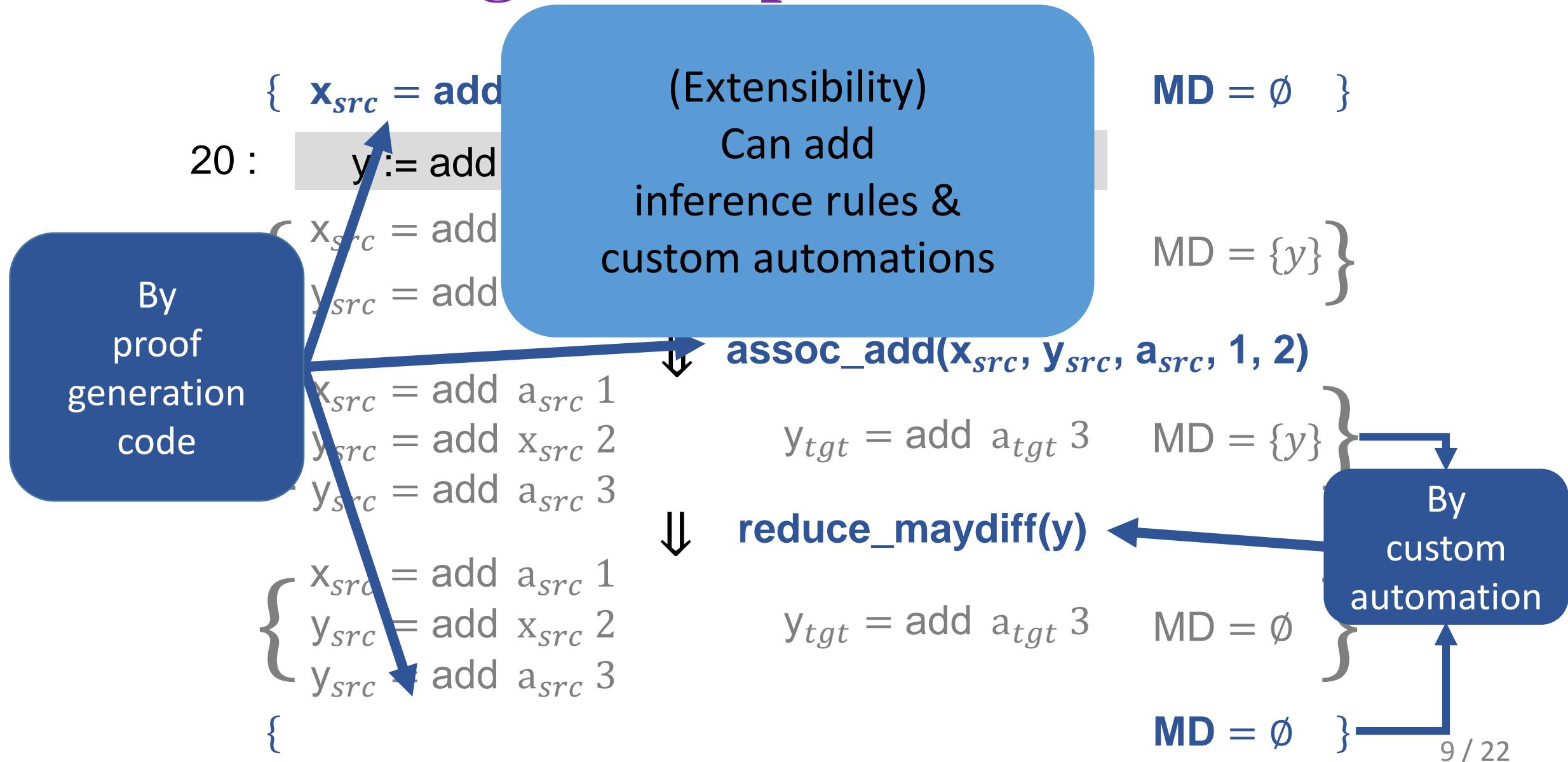
# ERHL: A Logic for Optimization Validation



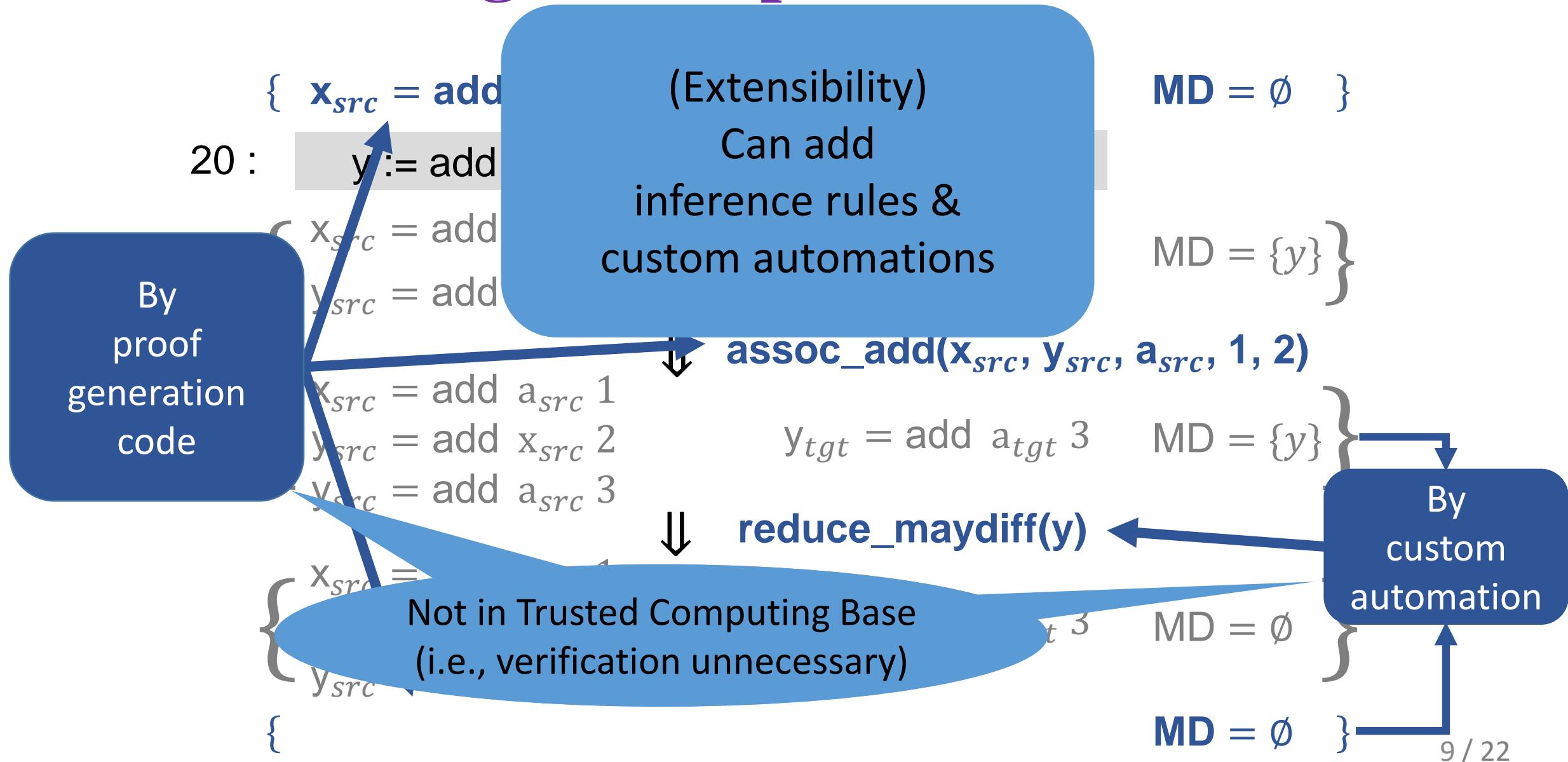
# ERHL: A Logic for Optimization Validation



# ERHL: A Logic for Optimization Validation



# ERHL: A Logic for Optimization Validation



# Proof Generation

---

**Algorithm 1** AssocAdd( $F$ : Function)

A1: **for**  $l_2: y := \text{add}(\text{reg } x) (\text{const } C_2)$  **in**  $F$  **do**

A2: **if** FindDef( $F, x$ ) **is**  $l_1: x := \text{add}(\text{reg } a) (\text{const } C_1)$  **then**

A3:  $C := \text{Simplify}(\text{add } C_1 \ C_2)$

A4: ReplaceAt( $F, l_2, y := \text{add}(\text{reg } a) (\text{const } C)$ )

A5:

A6:

A7: **end if**

A8: **end for**

A9:

10 :  $x := \text{add } a \ 1 \rightsquigarrow x := \text{add } a \ 1$

⋮

⋮

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

21 :  $\text{foo}(y) \rightsquigarrow \text{foo}(y)$

# Proof Generation

---

**Algorithm 1** AssocAdd( $F$ : Function)

A1: **for**  $l_2: y := \text{add}(\text{reg } x) (\text{const } C_2)$  **in**  $F$  **do**

A2: **if** FindDef( $F, x$ ) **is**  $l_1: x := \text{add}(\text{reg } a) (\text{const } C_1)$  **then**

A3:  $C := \text{Simplify}(\text{add } C_1 \ C_2)$

A4: ReplaceAt( $F, l_2, y := \text{add}(\text{reg } a) (\text{const } C)$ )

A5:

A6:

A7: **end if**

A8: **end for**

A9:

10 :  $x := \text{add } a \ 1 \rightsquigarrow x := \text{add } a \ 1$

⋮

⋮

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

21 :  $\text{foo}(y) \rightsquigarrow \text{foo}(y)$

# Proof Generation

---

**Algorithm 1** AssocAdd( $F$ : Function)

A1: **for**  $l_2$ :  $y := \text{add}(\text{reg } x, \text{const } C_2)$  **in**  $F$  **do**

A2: **if** FindDef( $F, x$ ) **is**  $l_1$ :  $x := \text{add}(\text{reg } a, \text{const } C_1)$  **then**

A3:  $C := \text{Simplify}(\text{add } C_1, C_2)$

A4: ReplaceAt( $F, l_2, y := \text{add}(\text{reg } a, \text{const } C)$ )

A5:

A6:

A7: **end if**

A8: **end for**

A9:

10:  $x := \text{add } a 1 \rightsquigarrow x := \text{add } a 1$

⋮ ⋮

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

21:  $\text{foo}(y) \rightsquigarrow \text{foo}(y)$

# Proof Generation

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

A1: **sum 1** AssocAdd( $F$ : Function)

10:  $x := \text{add } a \ 1$   $\rightsquigarrow x := \text{add } a \ 1$

A1: **if**  $l_2: y := \text{add } (\text{reg } x) (\text{const } C_2)$  **in**  $F$  **do**

⋮  
⋮

A2: **if** FindDef( $F, x$ ) **is**  $l_1: x := \text{add } (\text{reg } a) (\text{const } C_1)$  **then**

A3:  $C := \text{Simplify}(\text{add } C_1 \ C_2)$

A4: ReplaceAt( $F, l_2, y := \text{add } (\text{reg } a) (\text{const } C)$ )

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

A5:

A6:

A7: **end if**

A8: **end for**

A9:

21 :  $\text{foo}(y)$   $\rightsquigarrow \text{foo}(y)$

# Proof Generation

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

A1: **sum 1 AssocAdd( $F$ : Function)**

10:  $x := \text{add } a \ 1$   $\rightsquigarrow x := \text{add } a \ 1$

A1: **if**  $l_2: y := \text{add } (\text{reg } x) (\text{const } C_2)$  **in**  $F$  **do**

⋮  
⋮

A2: **if**  $\text{FindDef}(F, x)$  **is**  $l_1: x := \text{add } (\text{reg } a) (\text{const } C_1)$  **then**

A3:  $C := \text{Simplify}(\text{add } C_1 \ C_2)$

A4: ReplaceAt( $F, l_2, y := \text{add } (\text{reg } a) (\text{const } C)$ )

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

A5:

A6:

A7: **end if**

A8: **end for**

A9:

21 :  $\text{foo}(y)$   $\rightsquigarrow \text{foo}(y)$

# Proof Generation

---

**Algorithm 1** AssocAdd( $F$ : Function)

A1: **for**  $l_2$ :  $y := \text{add}(\text{reg } x, \text{const } C_2)$  **in**  $F$  **do**

A2: **if** FindDef( $F, x$ ) **is**  $l_1$ :  $x := \text{add}(\text{reg } a, \text{const } C_1)$  **then**

A3:  $C := \text{Simplify}(\text{add } C_1, C_2)$

A4: ReplaceAt( $F, l_2, y := \text{add}(\text{reg } a, \text{const } C)$ )

A5:

A6:

A7: **end if**

A8: **end for**

A9:

	{		$\text{MD} = \emptyset \}$
10:	$x := \text{add } a \ 1$	$\rightsquigarrow$	$x := \text{add } a \ 1$
	{		$\text{MD} = \emptyset \}$
	⋮	⋮	
	{		$\text{MD} = \emptyset \}$
20:	$y := \text{add } x \ 2$	$\rightsquigarrow$	$y := \text{add } a \ 3$
	{		$\text{MD} = \emptyset \}$
21 :	$\text{foo}(y)$	$\rightsquigarrow$	$\text{foo}(y)$
	{		$\text{MD} = \emptyset \}$

# Proof Generation

---

## Algorithm 1 AssocAdd( $F$ : Function)

---

A1: **for**  $l_2: y := \text{add}(\text{reg } x) (\text{const } C_2)$  **in**  $F$  **do**  
A2: **if**  $\text{FindDef}(F, x)$  **is**  $l_1: x := \text{add}(\text{reg } a) (\text{const } C_1)$  **then**  
A3:  $C := \text{Simplify}(\text{add } C_1 \ C_2)$   
A4:  $\text{ReplaceAt}(F, l_2, y := \text{add}(\text{reg } a) (\text{const } C))$

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

A6:

A7: **end if**

A8: **end for**

A9:

	{	$MD = \emptyset$ }
10:	$x := \text{add } a 1$ $\rightsquigarrow$ $x := \text{add } a 1$	$MD = \emptyset$ }
	{ $x_{src} = \text{add } a_{src} 1$ ⋮ $x_{src} = \text{add } a_{src} 1$ } ⋮	$MD = \emptyset$ }
20:	$y := \text{add } x 2$ $\rightsquigarrow$ $y := \text{add } a 3$	$MD = \emptyset$ }
	{ foo(y) $\rightsquigarrow$ foo(y) { }	$MD = \emptyset$ }

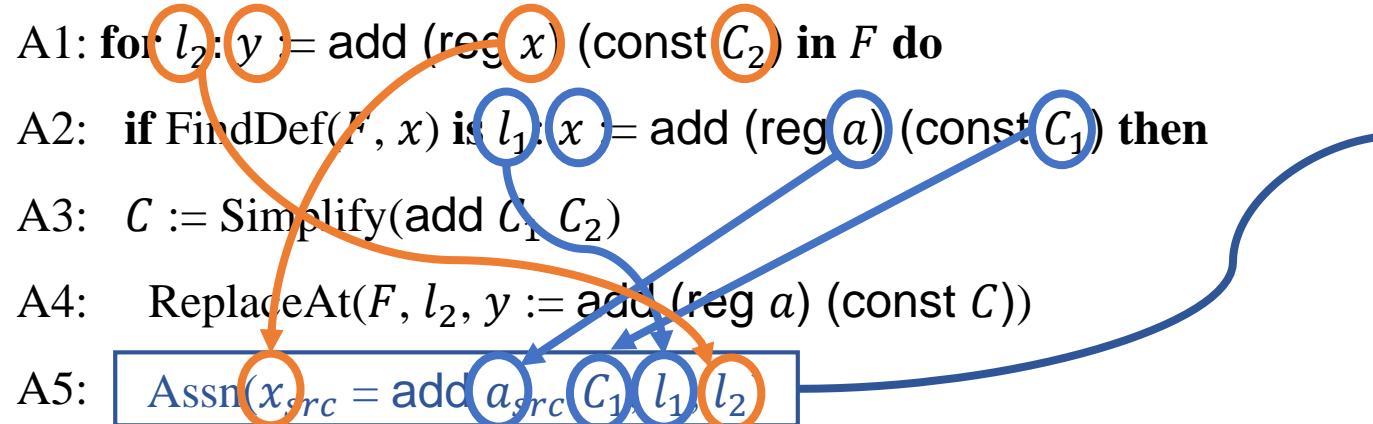
# Proof Generation

---

## Algorithm 1 AssocAdd( $F$ : Function)

---

A1: **for**  $l_2: y := \text{add}(\text{reg } x) (\text{const } C_2)$  **in**  $F$  **do**  
 A2: **if**  $\text{FindDef}(F, x)$  **is**  $l_1: x := \text{add}(\text{reg } a) (\text{const } C_1)$  **then**  
 A3:  $C := \text{Simplify}(\text{add } C_1 \ C_2)$   
 A4:  $\text{ReplaceAt}(F, l_2, y := \text{add}(\text{reg } a) (\text{const } C))$   
 A5: **Assn**( $x_{src} = \text{add } a_{src} \ C_1 \ l_1 \ l_2$ )



A6:

A7: **end if**

A8: **end for**

A9:

$\{$ <b>10:</b> $x := \text{add } a \ 1$ $\rightsquigarrow$ $x := \text{add } a \ 1$ $\{ \ x_{src} = \text{add } a_{src} \ 1$ $\vdots$ $\{ \ x_{src} = \text{add } a_{src} \ 1$ <b>20:</b> $y := \text{add } x \ 2$ $\rightsquigarrow$ $y := \text{add } a \ 3$	$\text{MD} = \emptyset \}$ $\text{MD} = \emptyset \}$ $\text{MD} = \emptyset \}$ $\text{MD} = \emptyset \}$
$\{$ <b>21 :</b> $\text{foo}(y)$ $\rightsquigarrow$ $\text{foo}(y)$ $\{$	$\text{MD} = \emptyset \}$ $\text{MD} = \emptyset \}$

# Proof Generation

---

## Algorithm 1 AssocAdd( $F$ : Function)

---

A1: **for**  $l_2$ :  $y := \text{add}(\text{reg } x, \text{const } C_2)$  **in**  $F$  **do**  
A2: **if**  $\text{FindDef}(F, x)$  **is**  $l_1$ :  $x := \text{add}(\text{reg } a, \text{const } C_1)$  **then**  
A3:  $C := \text{Simplify}(\text{add } C_1, C_2)$   
A4:  $\text{ReplaceAt}(F, l_2, y := \text{add}(\text{reg } a, \text{const } C))$   
A5:  $\text{Assn}(x_{src} = \text{add } a_{src}, C_1, l_1, l_2)$   
A6:  $\text{Inf}(\text{assoc\_add}(x_{src}, y_{src}, a_{src}, C_1, C_2), l_2)$   
A7: **end if**  
A8: **end for**  
A9:

	{	$MD = \emptyset$
10:	$x := \text{add } a 1 \rightsquigarrow x := \text{add } a 1$	$MD = \emptyset$
	{ $x_{src} = \text{add } a_{src} 1$	$MD = \emptyset$
	⋮	⋮
	{ $x_{src} = \text{add } a_{src} 1$	$MD = \emptyset$
20:	$y := \text{add } x 2 \rightsquigarrow y := \text{add } a 3$	$MD = \emptyset$
	<b>assoc_add(<math>x_{src}, y_{src}, a_{src}, 1, 2</math>)</b>	
	{	$MD = \emptyset$
21 :	$\text{foo}(y) \rightsquigarrow \text{foo}(y)$	$MD = \emptyset$
	{	$MD = \emptyset$

# Proof Generation

---

## Algorithm 1 AssocAdd( $F$ : Function)

---

A1: **for**  $l_2: y := \text{add}(\text{reg } x) (\text{const } C_2)$  **in**  $F$  **do**  
 A2: **if**  $\text{FindDef}(F, x)$  **is**  $l_1: x := \text{add}(\text{reg } a) (\text{const } C_1)$  **then**  
 A3:  $C := \text{Simplify}(\text{add } C_1, C_2)$   
 A4:  $\text{ReplaceAt}(F, l_2, y := \text{add}(\text{reg } a) (\text{const } C))$   
 A5:  $\text{Assn}(x_{src} = \text{add } a_{src} \ C_1, l_1, l_2)$   
 A6:  $\text{Inf}(\text{assoc\_add}(x_{src}, y_{src}, a_{src}, C_1, C_2), l_2)$   
 A7: **end if**  
 A8: **end for**  
 A9:

$\{$ <b>10:</b> <span style="border: 1px solid blue; padding: 2px;">x := add a 1</span> $\rightsquigarrow$ <span style="border: 1px solid gray; padding: 2px;">x := add a 1</span> $\{ x_{src} = \text{add } a_{src} 1$ <span style="background-color: #e0e0e0; display: inline-block; width: 100px; height: 15px;"></span> $\vdots$ <span style="background-color: #e0e0e0; display: inline-block; width: 100px; height: 15px;"></span> $\{ x_{src} = \text{add } a_{src} 1$ <b>20:</b> <span style="border: 1px solid blue; padding: 2px;">y := add x 2</span> $\rightsquigarrow$ <span style="border: 1px solid gray; padding: 2px;">y := add a 3</span> <span style="color: blue; font-weight: bold;">assoc_add(x<sub>src</sub>, y<sub>src</sub>, a<sub>src</sub>, 1, 2)</span> $\{$ <b>21:</b> <span style="border: 1px solid gray; padding: 2px;">foo(y)</span> $\rightsquigarrow$ <span style="border: 1px solid gray; padding: 2px;">foo(y)</span> $\{$	$\text{MD} = \emptyset \}$ $\text{MD} = \emptyset \}$
--	--

# Proof Generation

---

**Algorithm 1** AssocAdd( $F$ : Function)
 

---

A1: **for**  $l_2: y := \text{add}(\text{reg } x) (\text{const } C_2)$  **in**  $F$  **do**  
 A2: **if**  $\text{FindDef}(F, x)$  **is**  $l_1: x := \text{add}(\text{reg } a) (\text{const } C_1)$  **then**  
 A3:  $C := \text{Simplify}(\text{add } C_1 \ C_2)$   
 A4:  $\text{ReplaceAt}(F, l_2, y := \text{add}(\text{reg } a) (\text{const } C))$   
 A5:  $\text{Assn}(x_{src} = \text{add } a_{src} \ C_1, l_1, l_2)$   
 A6:  $\text{Inf}(\text{assoc\_add}(x_{src}, y_{src}, a_{src}, C_1, C_2), l_2)$   
 A7: **end if**  
 A8: **end for**  
 A9:  $\text{EnableAuto}(\text{reduce\_maydiff})$

10: <span style="border: 1px solid blue; padding: 2px;">{ <math>x := \text{add } a \ 1</math> }</span> $\rightsquigarrow$ <span style="border: 1px solid gray; padding: 2px;"><math>x := \text{add } a \ 1</math></span> $\quad \text{MD} = \emptyset$
<span style="border: 1px solid gray; padding: 2px;">{ <math>x_{src} = \text{add } a_{src} \ 1</math> } ⋮ { <math>x_{src} = \text{add } a_{src} \ 1</math> }</span> $\quad \text{MD} = \emptyset$
20: <span style="border: 1px solid blue; padding: 2px;">{ <math>y := \text{add } x \ 2</math> }</span> $\rightsquigarrow$ <span style="border: 1px solid gray; padding: 2px;"><math>y := \text{add } a \ 3</math></span> $\quad \text{MD} = \emptyset$
<span style="border: 1px solid blue; padding: 2px;"><b>assoc_add(<math>x_{src}, y_{src}, a_{src}, 1, 2</math>)</b></span>
<span style="border: 1px solid blue; padding: 2px;"><b>reduce_maydiff(<math>y</math>)</b></span>
21: <span style="border: 1px solid gray; padding: 2px;"><math>\text{foo}(y)</math></span> $\rightsquigarrow$ <span style="border: 1px solid gray; padding: 2px;"><math>\text{foo}(y)</math></span> $\quad \text{MD} = \emptyset$
<span style="border: 1px solid gray; padding: 2px;">{ }</span> $\quad \text{MD} = \emptyset$

# Proof Generation

---

## Algorithm 1 AssocAdd( $F$ : Function)

---

A1: **for**  $l_2: y := \text{add}(\text{reg } x) (\text{const } C_2)$  **in**  $F$  **do**  
A2: **if**  $\text{FindDef}(F, x)$  **is**  $l_1: x := \text{add}(\text{reg } a) (\text{const } C_1)$  **then**  
A3:  $C := \text{Simplify}(\text{add } C_1 \ C_2)$   
A4:  $\text{ReplaceAt}(F, l_2, y := \text{add}(\text{reg } a) (\text{const } C))$   
A5:  $\text{Assn}(x_{src} = \text{add } a_{src} \ C_1, l_1, l_2)$   
A6:  $\text{Inf}(\text{assoc\_add}(x_{src}, y_{src}, a_{src}, C_1, C_2), l_2)$   
A7: **end if**  
A8: **end for**  
A9:  $\text{EnableAuto}(\text{reduce\_maydiff})$

$\{$	$10 : x := \text{add } a \ 1 \rightsquigarrow x := \text{add } a \ 1$	$\text{MD} = \emptyset \}$
$\{ \ x_{src} = \text{add } a_{src} \ 1$	$\vdots$	$\text{MD} = \emptyset \}$
$\{ \ x_{src} = \text{add } a_{src} \ 1$	$\vdots$	$\text{MD} = \emptyset \}$
$= \text{add } x \ 2 \rightsquigarrow y := \text{add } a \ 3$	$\text{assoc\_add}(x_{src}, y_{src}, a_{src}, 1, 2)$	$\text{reduce\_maydiff}(y)$
$\{$	$21 : \text{foo}(y) \rightsquigarrow \text{foo}(y)$	$\text{MD} = \emptyset \}$
$\{$		$\text{MD} = \emptyset \}$

Extra code  
for Crellvm

# General Relational Properties in ERHL

- From the register promotion optimization

$$\left\{ \begin{array}{c} *p_{src} = r_{tgt} \\ \text{MD} = \{p, r\} \end{array} \right\}$$

# General Relational Properties in ERHL

- From the register promotion optimization



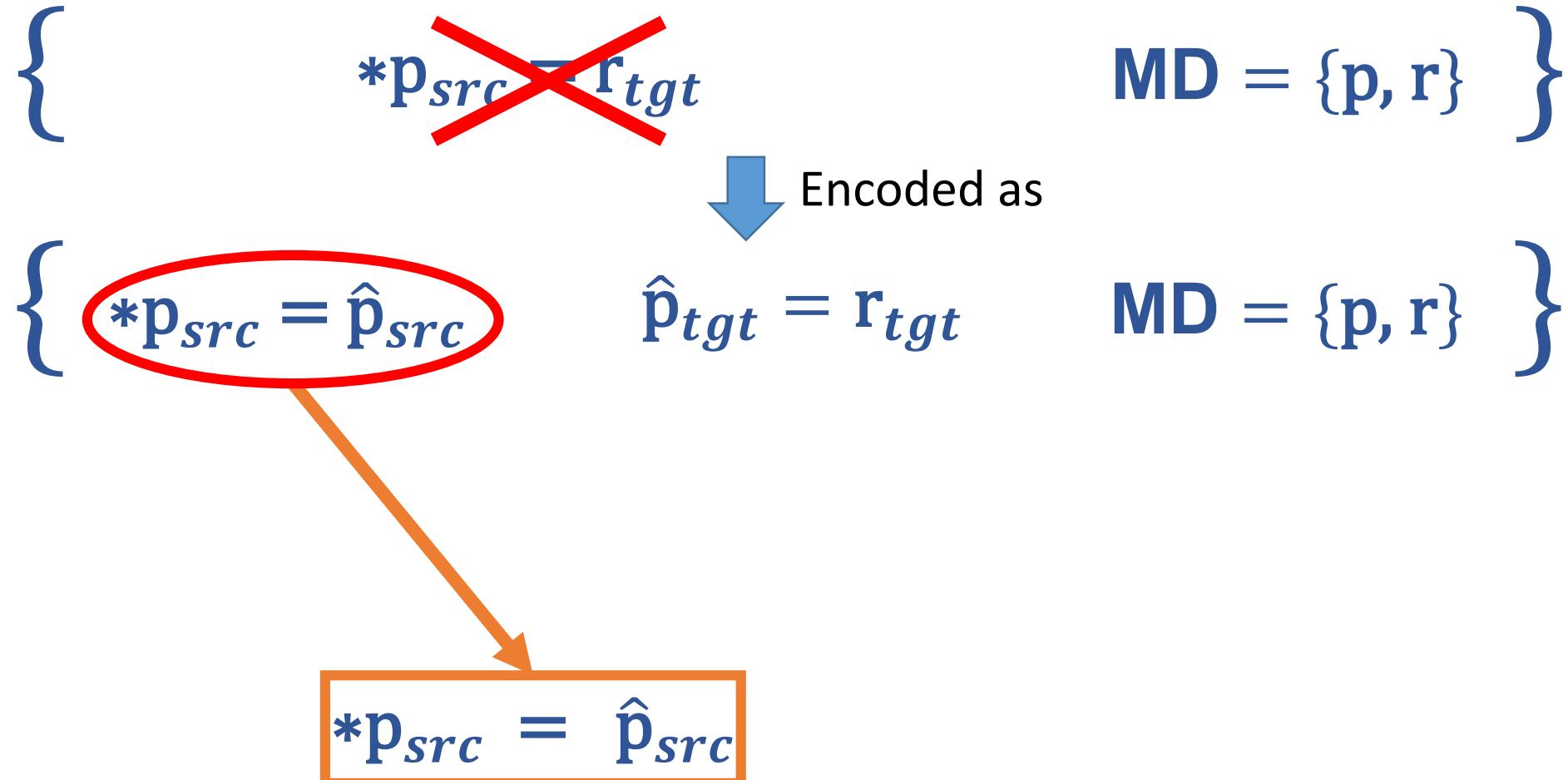
# General Relational Properties in ERHL

- From the register promotion optimization

$$\left\{ \begin{array}{c} \cancel{*p_{src} = r_{tgt}} \\ \downarrow \text{Encoded as} \\ *p_{src} = \hat{p}_{src} \quad \hat{p}_{tgt} = r_{tgt} \quad \mathbf{MD} = \{p, r\} \end{array} \right\}$$

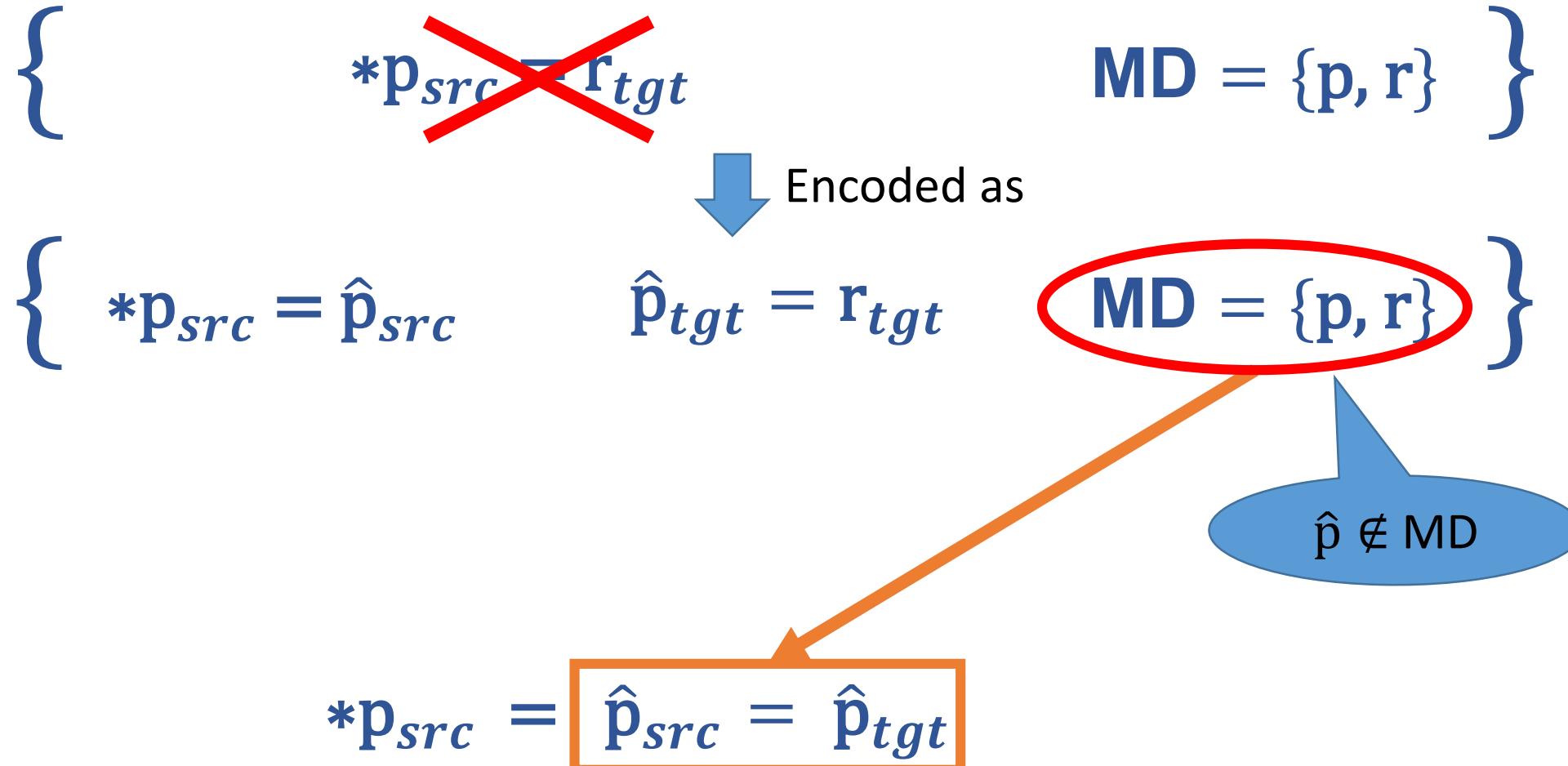
# General Relational Properties in ERHL

- From the register promotion optimization



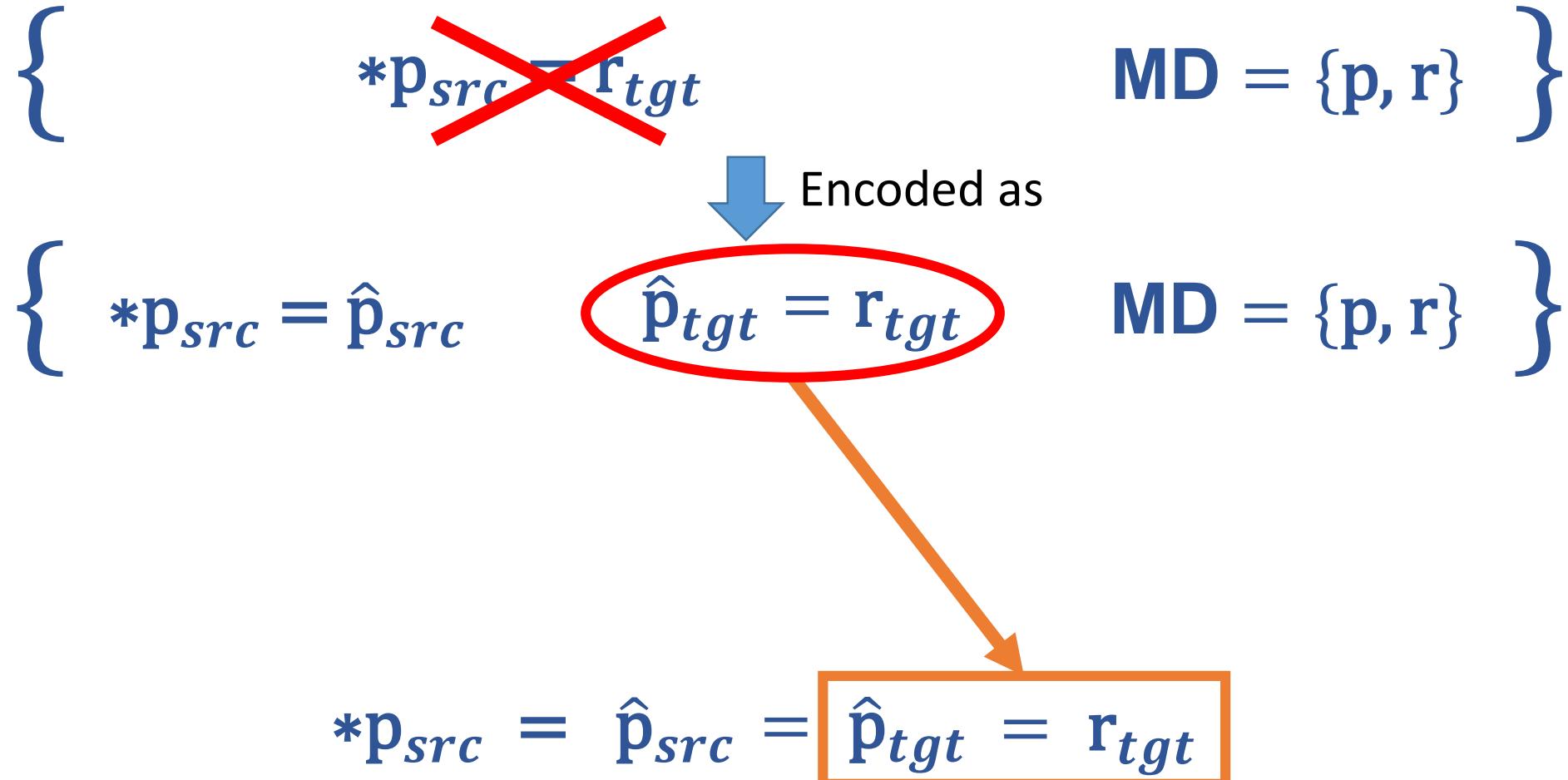
# General Relational Properties in ERHL

- From the register promotion optimization



# General Relational Properties in ERHL

- From the register promotion optimization



# General Relational Properties in ERHL

- From the register promotion optimization

$$\{ \quad \cancel{*p_{src} = r_{tgt}} \quad \text{MD} = \{p, r\} \}$$

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

Encoded as

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

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

# Implementation & Results

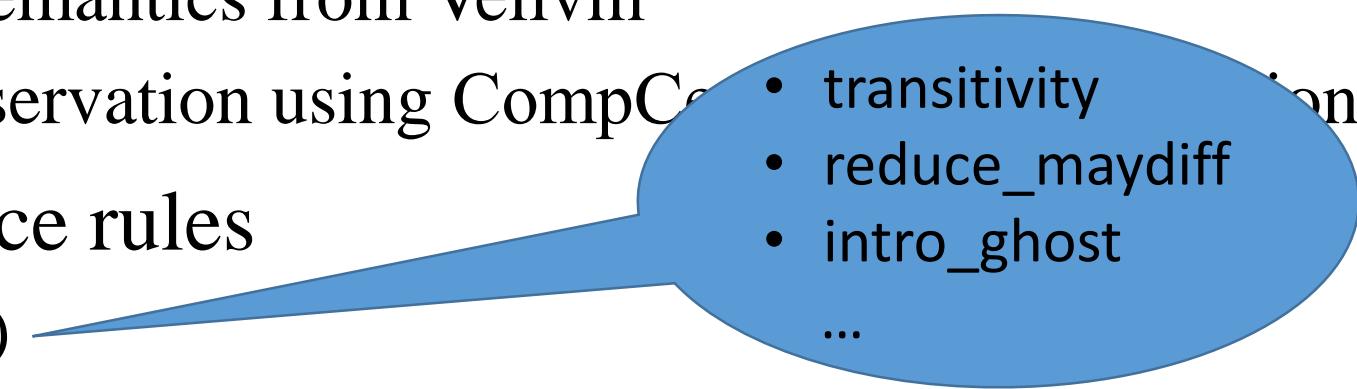
# Implementation: Proof Checker

- 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 (unverified)

---

	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 CompCert
  - Installed 221 inference rules
    - 9 main rules (verified)
    - 212 arithmetic & special rules for instcombine (unverified)
- 
- transitivity
  - reduce\_maydiff
  - intro\_ghost
  - ...

	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

	<code>mem2reg</code>	<code>gvn</code>	<code>licm</code>	<code>instcombine</code>
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