Better together: Unifying Datalog and Equality Saturation

Yihong Zhang, Remy Wang, Oliver Flatt, David Cao, Philip Zucker, Eli Rosenthal, Zach Tatlock, Max Willsey

PLDI 2023

Problem: we want everything

In Term Rewriting with EqSat	**In Program Analysis with Datalog**
+ Fast equational reasoning	+ Composable program analyses
- Poor analysis support	- Quadratic equational reasoning

Can we build one system that subsumes both?

Yes! But How?!

To unify Datalog and EqSat, all you need are

- Functional dependency.
- Functional dependency repair.

Background

EqSat: term rewriting with e-graphs

Big data systems

- Tensor programs [MLSys ‘21, MAPS ‘21]
- Sparse linear algebra [VLDB ‘20]
- Recursive queries [SIGMOD ‘22]

Hardware

- DSP vectorization [ASPLOS ‘23]
- Datapath optimization [ASP-DAC ‘23]

Program optimization

- Imperative programs [POPL ‘09]
- Functional programs [EGRAPHS ‘22]
- Floating-point expression [PLDI ‘15]

Program synthesis

- CAD parametrization [PLDI ‘20]
- Rewrite rule synthesis [OOPSLA ‘21]

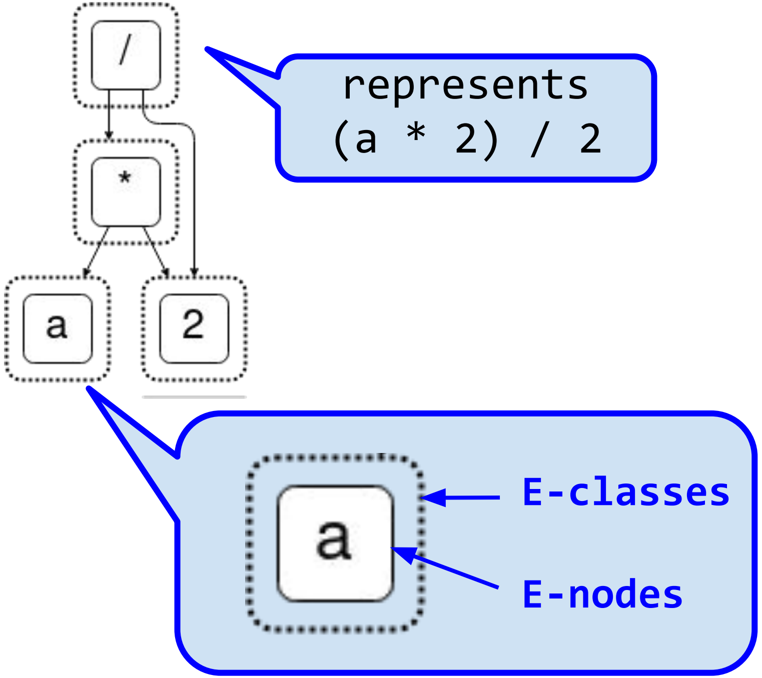
intel

RisingLight
An Educational OLAP Database System

CERTORA

BYTECODE
ALLIANCE

E-graphs and Equality saturation

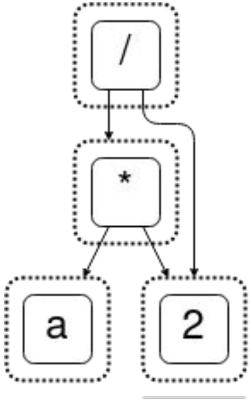


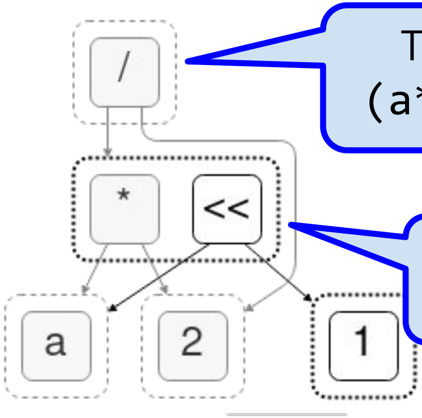
represents
(a * 2) / 2

E-classes

E-nodes

E-graphs and Equality saturation



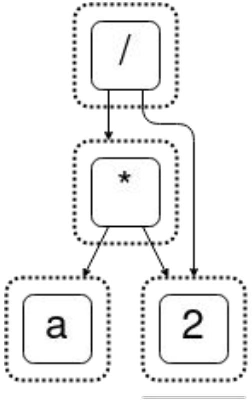


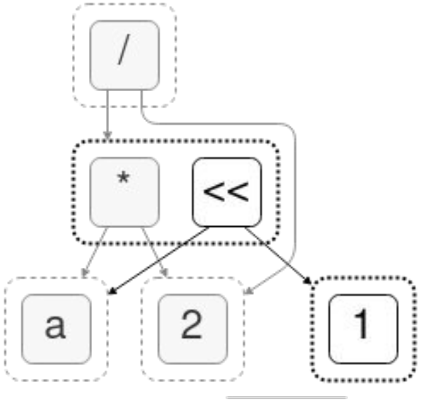
This e-class *represents* `(a*2)/2` **and** `(a<<1)/2`

This e-class *represents* `a * 2` **and** `a << 1`

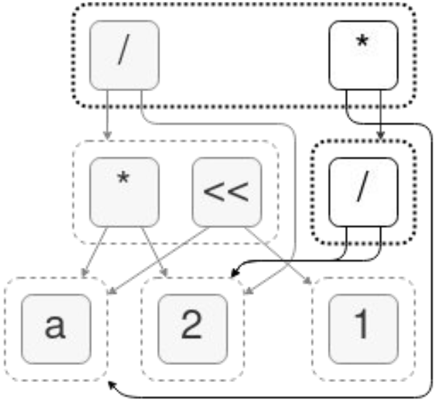
```
x * 2 => x << 1
```

E-graphs and Equality saturation



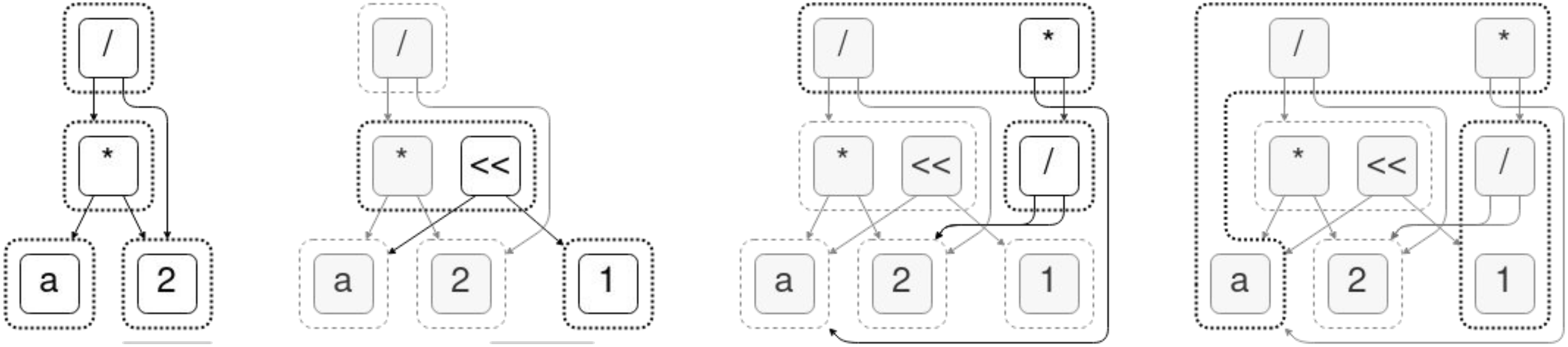


`x * 2 => x << 1`



`(x * y) / z => x * (y / z)`

E-graphs and Equality saturation



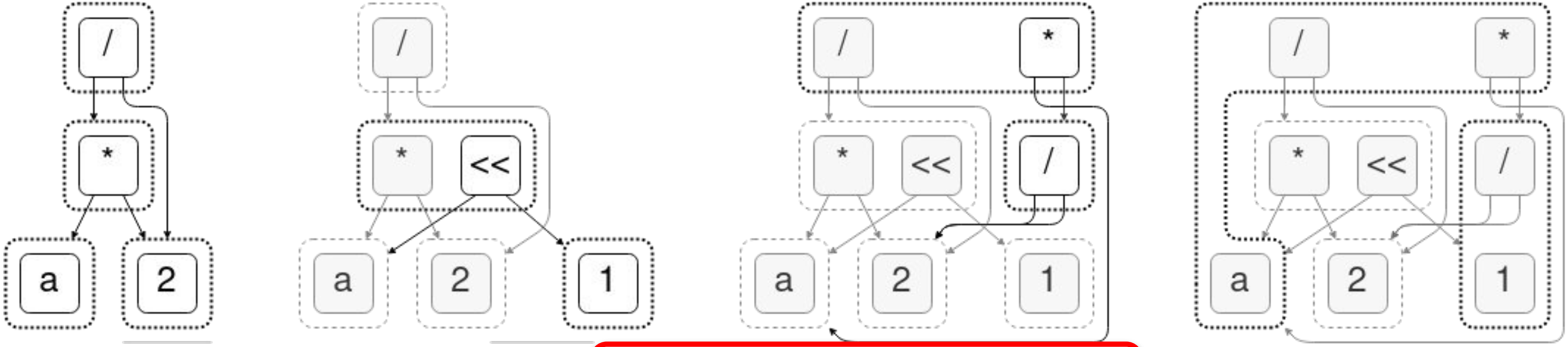
```
x * 2 => x << 1
```

```
(x * y) / z => x * (y / z)
```

```
x / x => 1
x * 1 => x
```

```
loop until
fixpoint / timeout!
```

E-graphs and Equality saturation



x * 2 => x <<

x has to be non-zero!

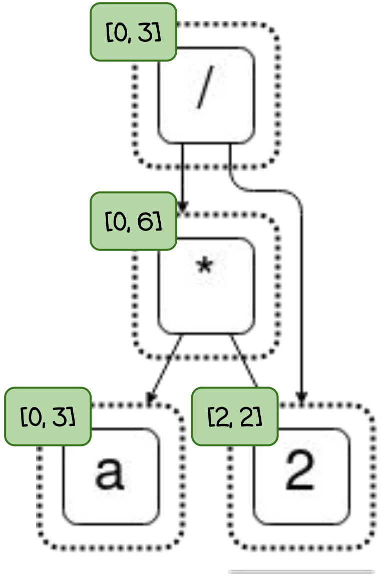
x / x => 1

x * 1 => x

loop until
xpoint / timeout!

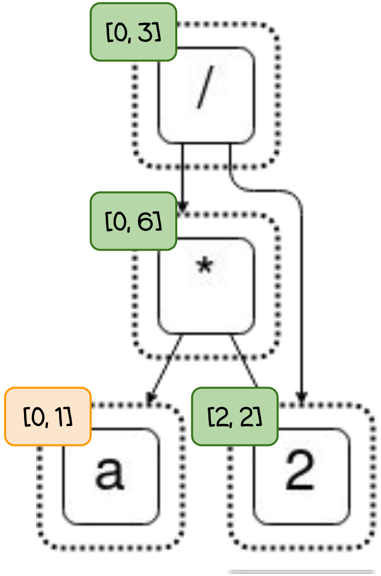
Needs semantic information here

E-class analyses



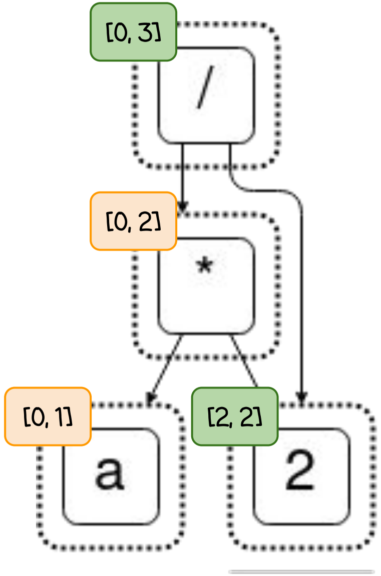
- Semantic analyses over E-graphs
- Each E-class is abstracted w/ a lattice.

E-class analyses



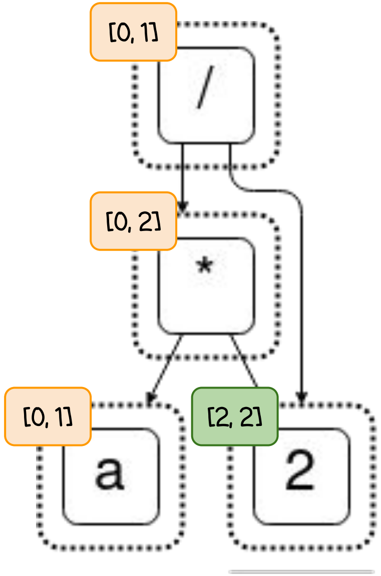
- Semantic analyses over E-graphs
- Each E-class is abstracted w/ a lattice.
- Information is propagated up.

E-class analyses



- Semantic analyses over E-graphs
- Each E-class is abstracted w/ a lattice.
- Information is propagated up.

E-class analyses



- Semantic analyses over E-graphs
- Each E-class is abstracted w/ a lattice.
- Information is propagated up.

E-class analyses

E-class analysis has severe limitations:

- Only one analysis allowed.
- Facts only propagate from children to parents.
  - Type checking 😭
- Monolithic Rust implementation of one big analysis.
  - Not composable!

tice.

BUT: analyses are rules, too!

Program analysis in Datalog

- Multiple analyses ✅
- Modular ✅
- Composable ✅

```
// If expression e is a number,
// its lower bound is itself
num(n, e) ⇒ lower_bound(e, n).

// If expression e has the form x + y,
// its lower bound is the lower bound of x
// plus the lower bound of y.
add(x, y, e) ∧
  lower_bound(x, lx) ∧
  lower_bound(y, ly) ⇒
lower_bound(e, lx + ly).

// If the lower bound of e is greater than 0,
// e is nonzero.
lower_bound(e, le) ∧ le > 0 ⇒
  nonzero(e)
```

Program analysis in Datalog

- Multiple analyses ✅
- Modular ✅
- Composable ✅

```
// If expression e is a number,
// its lower bound is itself
num(n, e) ⇒ lower_bound(e, n).

// If expression e has the form x + y,
// its lower bound is the lower bound of x
// plus the lower bound of y.
add(x, y, e) ∧
  lower_bound(x, lx) ∧
  lower_bound(y, ly) ⇒
lower_bound(e, lx + ly).

// If the lower bound of e is greater than 0,
// e is nonzero.
lower_bound(e, le) ∧ le > 0 ⇒
  nonzero(e)
```

Pr

```
// If expression e is a number
```

Can we do EqSat in Datalog?

```
f x
// plus the lower bound of y.
add(x, y, e) ∧
```

We need to express in Datalog

(+ (+ x y) z) → (+ x (+ y z))

triggers

actions

```
han 0,
```

Relational E-matching (POPL 2022)

- E-matching: pattern matching over the e-graph (triggers)

Relational E-matching (POPL 2022)

- E-matching: pattern matching over the e-graph (triggers)
- They are just database queries!

$$f(\alpha, g(\alpha)) \Rightarrow \begin{array}{c} Q(\alpha, root) \leftarrow \\ R_g(\alpha, x), R_f(\alpha, x, root) \end{array}$$

Relational E-matching (POPL 2022)

- E-matching: pattern matching over the e-graph (triggers)
- They are just database queries!
- Significant speedups.

$$f(\alpha, g(\alpha)) \Rightarrow \begin{array}{c} Q(\alpha, root) \leftarrow \\ R_g(\alpha, x), R_f(\alpha, x, root) \end{array}$$

Relational E-matching (POPL 2022)

- E-matching: pattern matching over the e-graph (triggers)
- They are just database queries!

$$f(\alpha, g(\alpha)) \Longrightarrow \begin{array}{c} Q(\alpha, root) \leftarrow \\ R_g(\alpha, x), R_f(\alpha, x, root) \end{array}$$

- Significant speedups.
- This handles triggers ✅

Relational E-matching (POPL 2022)

- E-matching: pattern matching over the e-graph (triggers)
- They are just database queries!

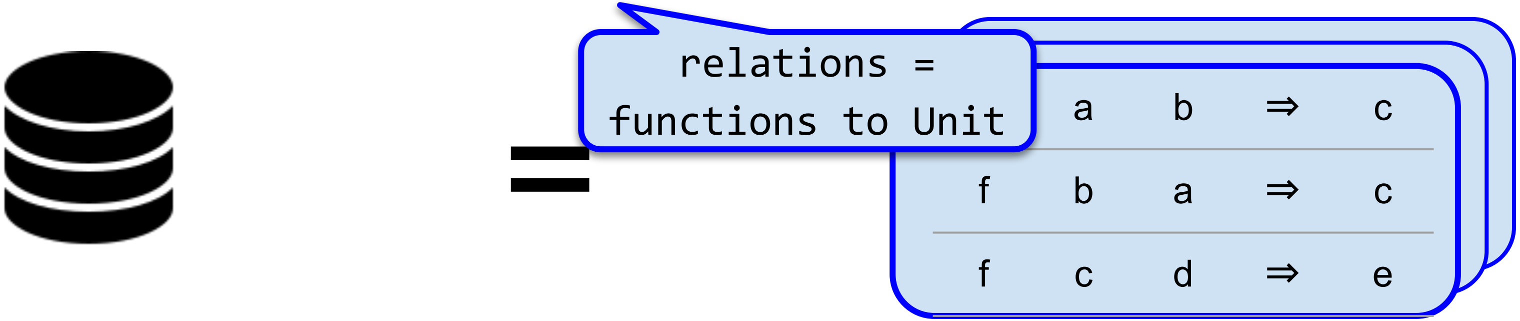
$$f(\alpha, g(\alpha)) \Longrightarrow \begin{array}{c} Q(\alpha, root) \leftarrow \\ R_g(\alpha, x), R_f(\alpha, x, root) \end{array}$$

- Significant speedups.
- This handles triggers ✅
- *What about actions?*

`egglog`: unifying Datalog and EqSat

egglog’s key concept: functions

Using a function-first database design



relations = functions to Unit

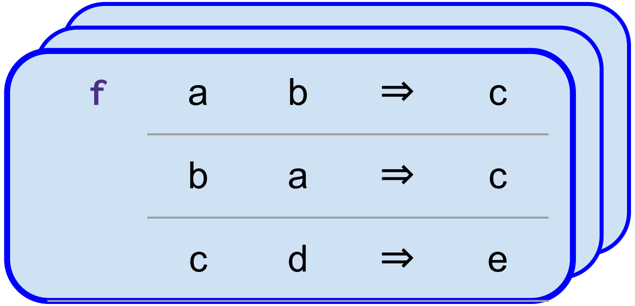
	a	b	⇒	c
f	b	a	⇒	c
f	c	d	⇒	e

egglog's key concept: functions

Using a function-first database design

Database

=



f	a	b	⇒	c
	b	a	⇒	c
	c	d	⇒	e

Set of function tables

Now we can talk about

- terms like `f(f(a, b), d)` and
- equivalences like `f(a, b)` = `f(b, a)`

Equality saturation in egglog

```
(datatype Math (Num i64)
               (Var String)
               (Add Math Math)
               (Mul Math Math))

;; expr = 3 * (x + 2)
(define expr (Mul (Num 3) (Add (Var "x") (Num 2))))

;; x + y => y + x
(rewrite (Add x y)         (Add y x))
;; x * (y + z) => x * y + x * z
(rewrite (Mul x (Add y z)) (Add (Mul x y) (Mul x z)))
;; Num(x) + Num(y) => Num(x + y)
(rewrite (Add (Num x) (Num y)) (Num (+ x y)))
;; Num(x) * Num(y) => Num(x * y)
(rewrite (Mul (Num x) (Num y)) (Num (* x y)))
```

Equality saturation in egglog

```
(datatype Math (Num i64)
               (Var String)
               (Add Math Math)
               (Mul Math Math))

;; expr = 3 * (x + 2)
(define expr (Mul (Num 3) (Add (Var "x") (Num 2))))

;; x + y => y + x
(rewrite (Add x y)         (Add y x))
;; x * (y + z) => x * y + x * z
(rewrite (Mul x (Add y z)) (Add (Mul x y) (Mul x z)))
;; Num(x) + Num(y) => Num(x + y)
(rewrite (Add (Num x) (Num y)) (Num (+ x y)))
;; Num(x) * Num(y) => Num(x * y)
(rewrite (Mul (Num x) (Num y)) (Num (* x y)))
```

Num	i64		⇒	Math
Var	String		⇒	Math
Add	Math	Math	⇒	Math
Mul	Math	Math	⇒	Math

Equality saturation in egglog

```
(datatype Math (Num i64)
               (Var String)
               (Add Math Math)
               (Mul Math Math))

;; expr = 3 * (x + 2)
(define expr (Mul (Num 3) (Add (Var "x") (Num 2))))

;; x + y => y + x
(rewrite (Add x y)         (Add y x))
;; x * (y + z) => x * y + x * z
(rewrite (Mul x (Add y z)) (Add (Mul x y) (Mul x z)))
;; Num(x) + Num(y) => Num(x + y)
(rewrite (Add (Num x) (Num y)) (Num (+ x y)))
;; Num(x) * Num(y) => Num(x * y)
(rewrite (Mul (Num x) (Num y)) (Num (* x y)))
```

Num	2	⇒	C_1	
	3	⇒	C_2	
Var	"x"	⇒	C_3	
Add	C_3	C_1	⇒	C_4
Mul	C_2	C_4	⇒	C_5

Equality saturation in egglog

```
(datatype Math (Num i64)
               (Var String)
               (Add Math Math)
               (Mul Math Math))

;; expr = 3 * (x + 2)
(define expr (Mul (Num 3) (Add (Var "x") (Num 2))))
```

```
;; x + y => y + x
(rewrite (Add x y)         (Add y x))
;; x * (y + z) => x * y + x * z
(rewrite (Mul x (Add y z)) (Add (Mul x y) (Mul x z)))
;; Num(x) + Num(y) => Num(x + y)
(rewrite (Add (Num x) (Num y)) (Num (+ x y)))
;; Num(x) * Num(y) => Num(x * y)
(rewrite (Mul (Num x) (Num y)) (Num (* x y)))
```

Num	2	⇒	c_1	
	3	⇒	c_2	
Var	"x"	⇒	c_3	
Add	c_3	c_1	⇒	c_4
Mul	c_2	c_4	⇒	c_5

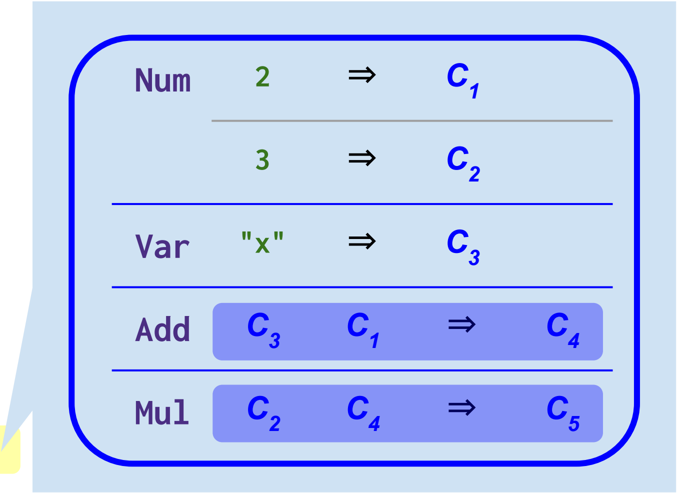
Equality saturation in egglog

```
(datatype Math (Num i64)
               (Var String)
               (Add Math Math)
               (Mul Math Math))

;; expr = 3 * (x + 2)
(define expr (Mul (Num 3) (Add (Var "x") (Num 2))))

;; x + y => y + x
(rewrite (Add x y)         (Add y x))
;; x * (y + z) => x * y + x * z
(rewrite (Mul x (Add y z)) (Add (Mul x y) (Mul x z)))
;; Num(x) + Num(y) => Num(x + y)
(rewrite (Add (Num x) (Num y)) (Num (+ x y)))
;; Num(x) * Num(y) => Num(x * y)
(rewrite (Mul (Num x) (Num y)) (Num (* x y)))
```

Num	2	⇒	C_1	
	3	⇒	C_2	
Var	"x"	⇒	C_3	
Add	C_3	C_1	⇒	C_4
Mul	C_2	C_4	⇒	C_5

Equality saturation in egglog

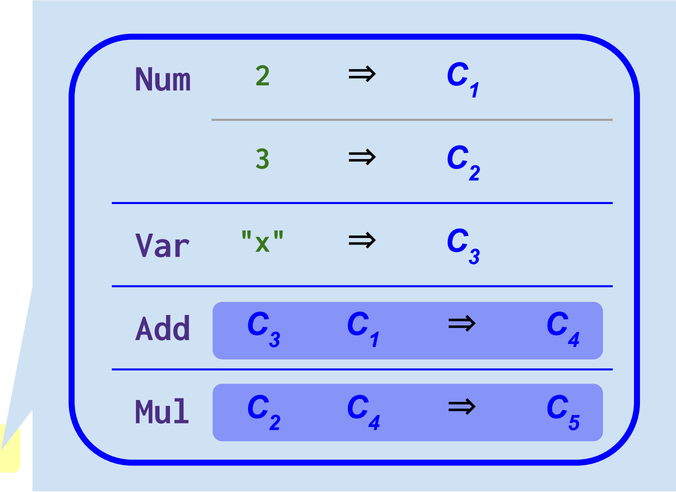
```
(datatype Math (Num i64)
               (Var String)
               (Add Math Math)
               (Mul Math Math))

;; expr = 3 * (x + 2)
(define expr (Mul (Num 3) (Add (Var "x") (Num 2))))

;; x + y => y + x
(rewrite (Add x y)         (Add y x))
;; x * (y + z) => x * y + x * z
(rewrite (Mul x (Add y z)) (Add (Mul x y) (Mul x z)))
;; Num(x) + Num(y) => Num(x + y)
(rewrite (Add (Num x) (Num y)) (Num (+ x y)))
;; Num(x) * Num(y) => Num(x * y)
(rewrite (Mul (Num x) (Num y)) (Num (* x y)))
```



Num	2		⇒	c_1
	3		⇒	c_2
Var	"x"		⇒	c_3
Add	c_3	c_1	⇒	c_4
Mul	c_2	c_4	⇒	c_5

Equality saturation in egglog

```
(datatype Math (Num i64)
               (Var String)
               (Add Math Math)
               (Mul Math Math))

;; expr = 3 * (x + 2)
(define expr (Mul (Num 3) (Add (Var "x") (Num 2))))

;; x + y => y + x
(rewrite (Add x y)         (Add y x))
;; x * (y + z) => x * y + x * z
(rewrite (Mul x (Add y z)) (Add (Mul x y) (Mul x z)))
;; Num(x) + Num(y) => Num(x + y)
(rewrite (Add (Num x) (Num y)) (Num (+ x y)))
;; Num(x) * Num(y) => Num(x * y)
(rewrite (Mul (Num x) (Num y)) (Num (* x y)))
```



Num	2		⇒	C_1
	3		⇒	C_2
Var	"x"		⇒	C_3
Add	C_3	C_1	⇒	C_4
Mul	C_2	C_4	⇒	C_5

Equality saturation in egglog

```
(datatype Math (Num i64)
               (Var String)
               (Add Math Math)
               (Mul Math Math))

;; expr = 3 * (x + 2)
(define expr (Mul (Nu            r "x") (Num 2))))

;; x + y => y + x
(rewrite (Add x y)             x))
;; x * (y + z) => x *   y + x * z
(rewrite (Mul x (Add y z)) (Add (Mul x y) (Mul x z)))
;; Num(x) + Num(y) => Num(x + y)
(rewrite (Add (Num x) (Num y)) (Num (+ x y)))
;; Num(x) * Num(y) => Num(x * y)
(rewrite (Mul (Num x) (Num y)) (Num (* x y)))
```

{x ↦ c_2,
y ↦ c_3,
z ↦ c_1}

Num	2		⇒	c_1
	3		⇒	c_2
Var	"x"		⇒	c_3
Add	c_3	c_1	⇒	c_4
Mul	c_2	c_4	⇒	c_5

Equality saturation in egglog

```
(datatype Math (Num i64)
               (Var String)
               (Add Math Math)
               (Mul Math Math))

;; expr = 3 * (x + 2)
(define expr (Mul (Nu            r "x") (Num 2))))

;; x + y => y + x
(rewrite (Add x y)              x))
;; x * (y + z) => x * y + x * z
(rewrite (Mul x (Add y z)) (Add (Mul x y) (Mul x z)))
;; Num(x) + Num(y) => Num(x + y)
(rewrite (Add (Num x) (Num y)) (Num (+ x y)))
;; Num(x) * Num(y) => Num(x * y)
(rewrite (Mul (Num x) (Num y)) (Num (* x y)))
```

$$\{x \mapsto C_2,\; y \mapsto C_3,\; z \mapsto C_1\}$$

Num	2		⇒	C_1
	3		⇒	C_2
Var	"x"		⇒	C_3
Add	C_3	C_1	⇒	C_4
Mul	C_2	C_4	⇒	C_5

Equality saturation in egglog

```
(datatype Math (Num i64)
               (Var String)
               (Add Math Math)
               (Mul Math Math))

;; expr = 3 * (x + 2)
(define expr (Mul (Nu            r "x") (Num 2))))

;; x + y => y + x
(rewrite (Add x y)              x))
;; x * (y + z) => x * y + x * z
(rewrite (Mul x (Add y z)) (Add (Mul x y) (Mul x z)))
;; Num(x) + Num(y) => Num(x + y)
(rewrite (Add (Num x) (Num y)) (Num (+ x y)))
;; Num(x) * Num(y) => Num(x * y)
(rewrite (Mul (Num x) (Num y)) (Num (* x y)))
```

$\{x \mapsto c_2,$
$y \mapsto c_3,$
$z \mapsto c_1\}$

Num	2	⇒	c_1	
	3	⇒	c_2	
Var	"x"	⇒	c_3	
Add	c_3	c_1	⇒	c_4
Mul	c_4	c_2	⇒	c_5
	c_2	c_3	⇒	c_6
	c_2	c_1	⇒	c_7

Equality saturation in egglog

```
(datatype Math (Num i64)
               (Var String)
               (Add Math Math)
               (Mul Math Math))

;; expr = 3 * (x + 2)
(define expr (Mul (Nu            r "x") (Num 2))))

;; x + y => y + x
(rewrite (Add x y)               x))
;; x * (y + z) => x * y + x * z
(rewrite (Mul x (Add y z)) (Add (Mul x y) (Mul x z)))
;; Num(x) + Num(y) => Num(x + y)
(rewrite (Add (Num x) (Num y)) (Num (+ x y)))
;; Num(x) * Num(y) => Num(x * y)
(rewrite (Mul (Num x) (Num y)) (Num (* x y)))
```

$\{x \mapsto c_2,$
$y \mapsto c_3,$
$z \mapsto c_1\}$

Num	2	⇒	c_1	
	3	⇒	c_2	
Var	"x"	⇒	c_3	
Add	c_3	c_1	⇒	c_4
	c_6	c_7	⇒	c_8
Mul	c_4	c_2	⇒	c_5
	c_2	c_3	⇒	c_6
	c_2	c_1	⇒	c_7

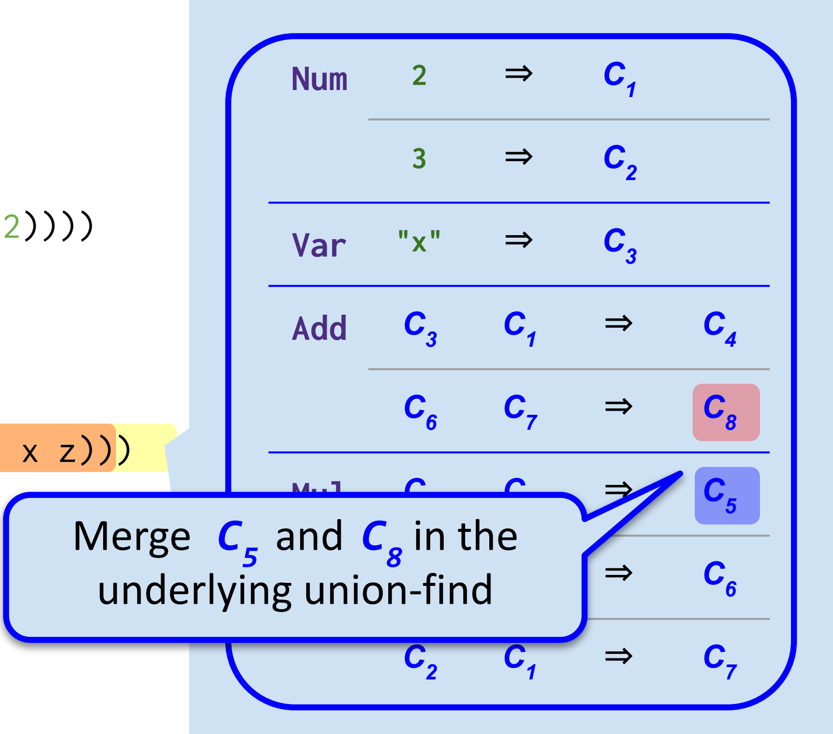
Equality saturation in egglog

```
(datatype Math (Num i64)
               (Var String)
               (Add Math Math)
               (Mul Math Math))

;; expr = 3 * (x + 2)
(define expr (Mul (Num 3) (Add (Var "x") (Num 2))))

;; x + y => y + x
(rewrite (Add x y)         (Add y x))
;; x * (y + z) => x * y + x * z
(rewrite (Mul x (Add y z)) (Add (Mul x y) (Mul x z)))
;; Num(x) + Num(y) => Num(x + y)
(rewrite (Add (Num x) (Num y)) (Num (+ x y)))
;; Num(x) * Num(y) => Num(x * y)
(rewrite (Mul (Num x) (Num y)) (Num (* x y)))
```

Num	2		⇒	c_1
	3		⇒	c_2
Var	"x"		⇒	c_3
Add	c_3	c_1	⇒	c_4
	c_6	c_7	⇒	c_8
Mul	c_4	c_2	⇒	c_5
	c_2	c_3	⇒	c_6
	c_2	c_1	⇒	c_7

Equality saturation in egglog

```
(datatype Math (Num i64)
               (Var String)
               (Add Math Math)
               (Mul Math Math))

;; expr = 3 * (x + 2)
(define expr (Mul (Num 3) (Add (Var "x") (Num 2))))

;; x + y => y + x
(rewrite (Add x y)         (Add y x))
;; x * (y + z) => x * y + x * z
(rewrite (Mul x (Add y z)) (Add (Mul x y) (Mul x z)))
;; Num(x) + Num(y) => Num(x + y)
(rewrite (Add (Num x) (Num y)) (Num (+ x y)))
;; Num(x) * Num(y) => Num(x * y)
(rewrite (Mul (Num x) (Num y)) (Num (* x y)))
```

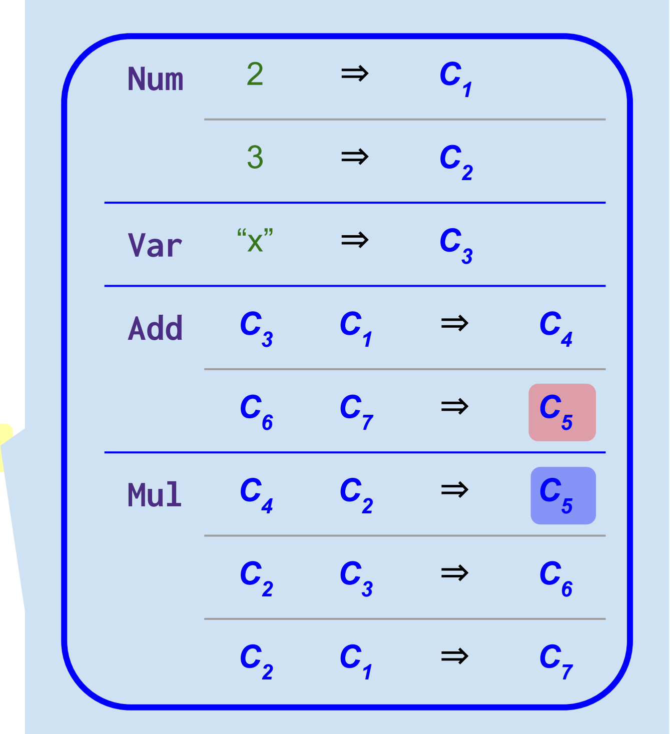


Num	2	⇒	C_1	
	3	⇒	C_2	
Var	"x"	⇒	C_3	
Add	C_3	C_1	⇒	C_4
	C_6	C_7	⇒	C_8
Mul	C	C	⇒	C_5
			⇒	C_6
	C_2	C_1	⇒	C_7

Merge C_5 and C_8 in the underlying union-find

Equality saturation in egglog

```
(datatype Math (Num i64)
               (Var String)
               (Add Math Math)
               (Mul Math Math))

;; expr = 3 * (x + 2)
(define expr (Mul (Num 3) (Add (Var "x") (Num 2))))

;; x + y => y + x
(rewrite (Add x y)         (Add y x))
;; x * (y + z) => x * y + x * z
(rewrite (Mul x (Add y z)) (Add (Mul x y) (Mul x z)))
;; Num(x) + Num(y) => Num(x + y)
(rewrite (Add (Num x) (Num y)) (Num (+ x y)))
;; Num(x) * Num(y) => Num(x * y)
(rewrite (Mul (Num x) (Num y)) (Num (* x y)))
```



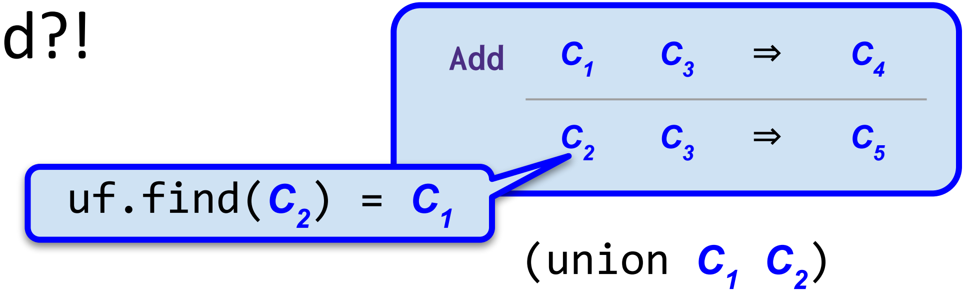
Num	2		⇒	C_1
	3		⇒	C_2
Var	“x”		⇒	C_3
Add	C_3	C_1	⇒	C_4
	C_6	C_7	⇒	C_5
Mul	C_4	C_2	⇒	C_5
	C_2	C_3	⇒	C_6
	C_2	C_1	⇒	C_7

Key idea: functional dependency repair

- Func’s args should uniquely determine the output.
- This is what makes a function a function.
- What if this is violated?!

Key idea: functional dependency repair

- Func’s args should uniquely determine the output.
- This is what makes a function a function.
- What if this is violated?!



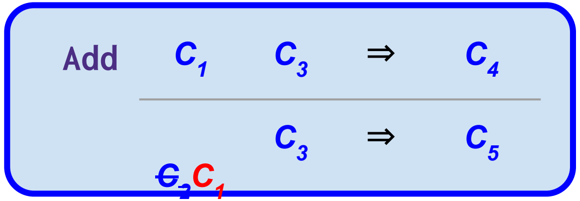
Add	C_1	C_3	⇒	C_4
	C_2	C_3	⇒	C_5

uf.find(C_2) = C_1

(union C_1 C_2)

Key idea: functional dependency repair

- Func's args should uniquely determine the output.
- This is what makes a function a function.
- What if this is violated?!



Add	C_1	C_3	⇒	C_4
	~~C_2~~ C_1	C_3	⇒	C_5

(union C_1 C_2)

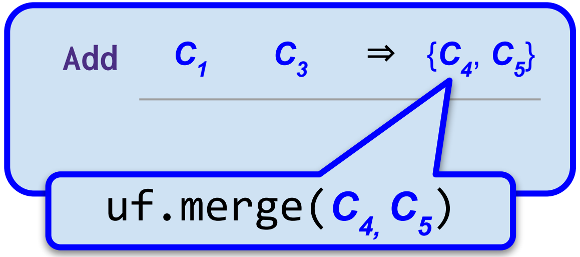
Key idea: functional dependency repair

- Func’s args should uniquely determine the output.
- This is what makes a function a function.
- What if this is violated?!

Add C_1 C_3 ⇒ $\{C_4, C_5\}$

Key idea: functional dependency repair

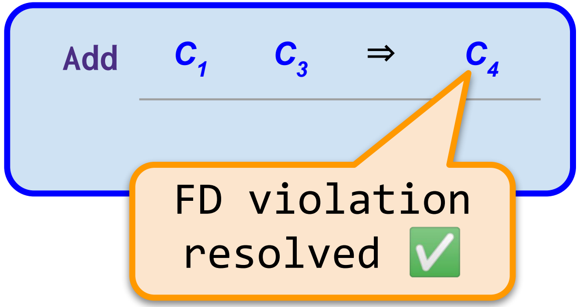
- Func's args should uniquely determine the output.
- This is what makes a function a function.
- What if this is violated?!
- We merge the conflicting values with a union find!



```
uf.merge(C4, C5)
```

Key idea: functional dependency repair

- Func’s args should uniquely determine the output.
- This is what makes a function a function.
- What if this is violated?!
- We merge the conflicting values with a union find!



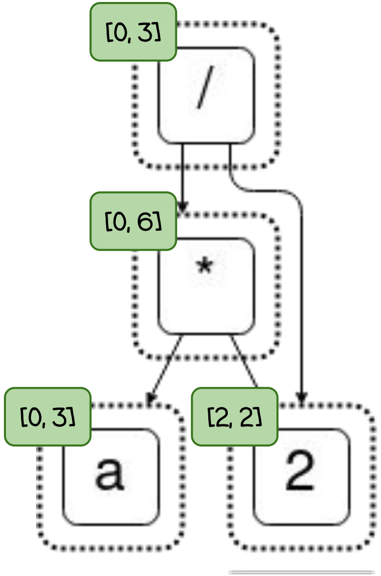
Add c_1 c_3 ⇒ c_4

FD violation resolved ✅

Key idea: functional dependency repair

The same mechanism also enables **composable** analyses.

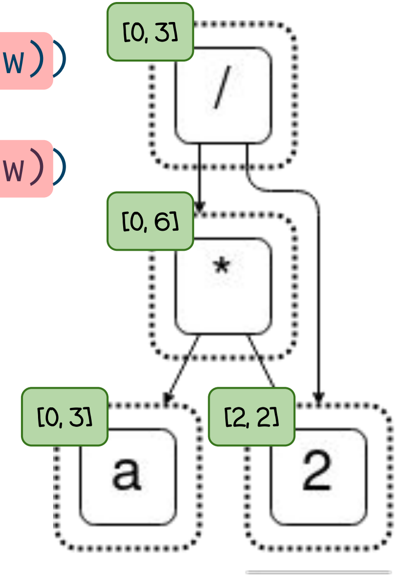
```
(function hi (Math) Rational)

(function lo (Math) Rational)
```



Key idea: functional dependency repair

The same mechanism also enables **composable** analyses.

```
(function hi (Math) Rational :merge (min old new))

(function lo (Math) Rational :merge (max old new))
```



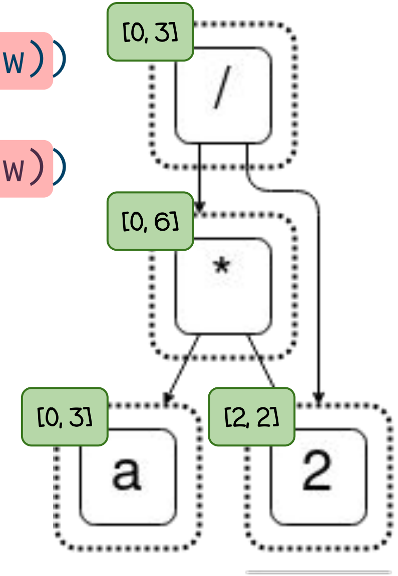
Key idea: functional dependency repair

The same mechanism also enables **composable** analyses.

```
(function hi (Math) Rational :merge (min old new))

(function lo (Math) Rational :merge (max old new))
```

hi	c_1	⇒	3
	c_2	⇒	2

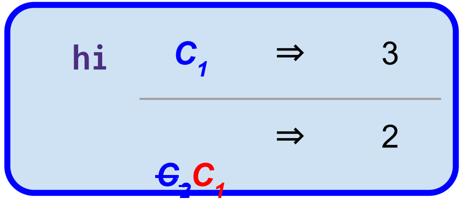
(union c_1 c_2)



Key idea: functional dependency repair

The same mechanism also enables **composable** analyses.

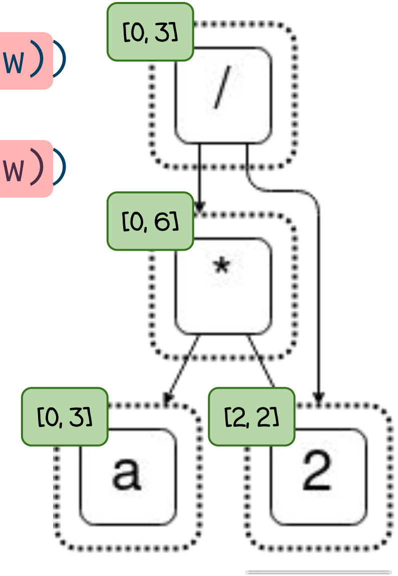
```
(function hi (Math) Rational :merge (min old new))

(function lo (Math) Rational :merge (max old new))
```



hi C_1 ⇒ 3

~~C_2~~ C_1 ⇒ 2

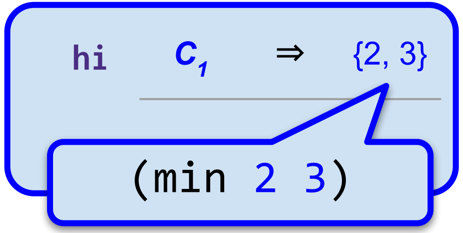
(union C_1 C_2)

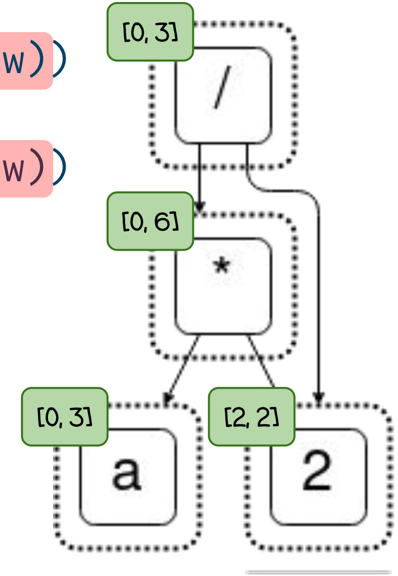


Key idea: functional dependency repair

The same mechanism also enables **composable** analyses.

```
(function hi (Math) Rational :merge (min old new))

(function lo (Math) Rational :merge (max old new))
```





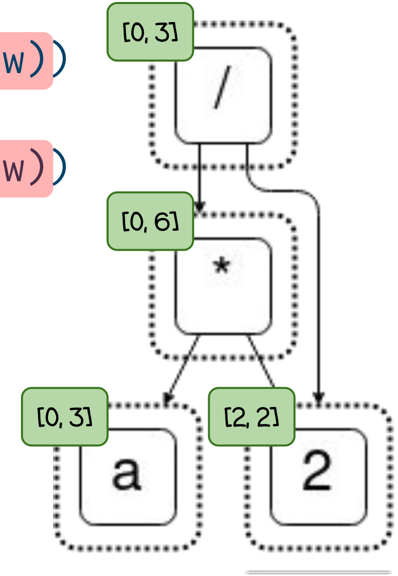
Key idea: functional dependency repair

The same mechanism also enables **composable** analyses.

```
(function hi (Math) Rational :merge (min old new))

(function lo (Math) Rational :merge (max old new))
```

hi C_1 ⇒ 2

FD violation resolved ✅



Evaluation

Enabling new optimizations

Database-like architecture

- Relational e-matching.
- Efficient query evaluation.

Enabling new optimizations

Database-like architecture

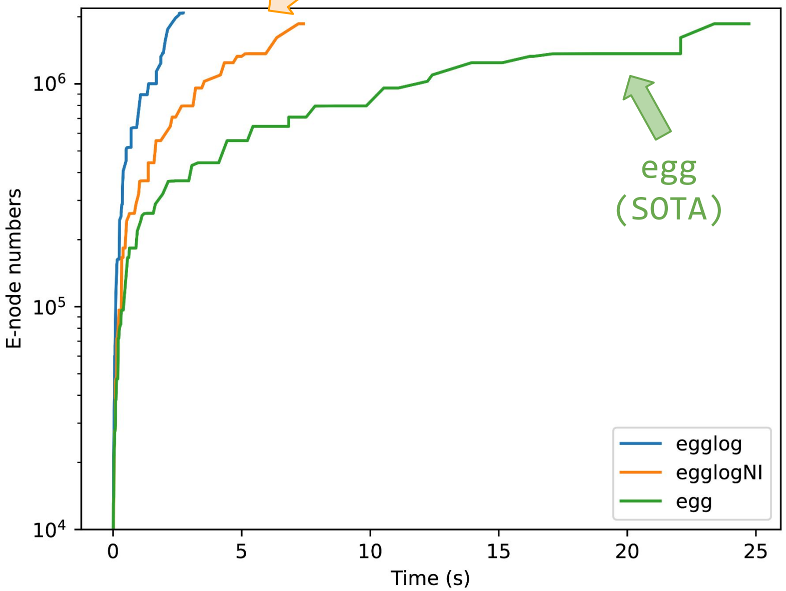
- Relational e-matching.
- Efficient query evaluation.

Incrementalization

- Incremental EqSat is hard.
- We use the standard semi-naive evaluation of Datalog to make EqSat incremental for free.

egglog (9.27×)

non-incremental egglog (3.34×)



egg (SOTA)

math micro-benchmark

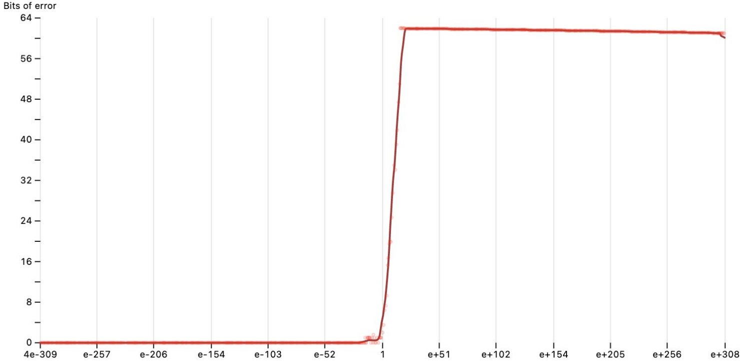
Herbie

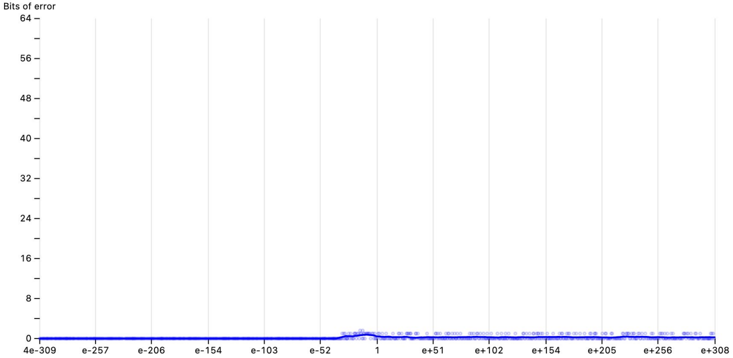
$$\sqrt{x+1}-\sqrt{x}$$

→

$$\frac{1}{\sqrt{1+x}+\sqrt{x}}$$

less floating-point errors,
more accurate!

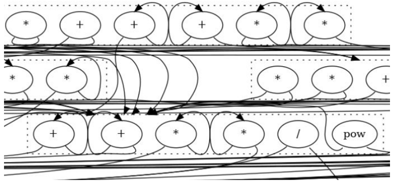




Herbie

$$\sqrt{x+1} - \sqrt{x}$$

EqSat



Extract

$$\frac{1}{\sqrt{1+x}+\sqrt{x}}$$

$x + y \Rightarrow y + x$

$(x + y) + z \Rightarrow x + (y + z)$

$\sin^2 x + \cos^2 x \Rightarrow 1$

$x^{yz} \Rightarrow x^y \cdot x^z$

$x^{-1} \Rightarrow 1 / x$

…

Not all identities can be expressed as syntactic rules.

$x / x \Rightarrow 1$ **when** $x \neq 0$

$(x \cdot y) / z \Rightarrow x / (z / y)$
when $y \neq 0$

$x - y \Rightarrow (x^3 - y^3) / (x^2 + xy + y^2)$
when $x \neq 0$ **or** $y \neq 0$

…

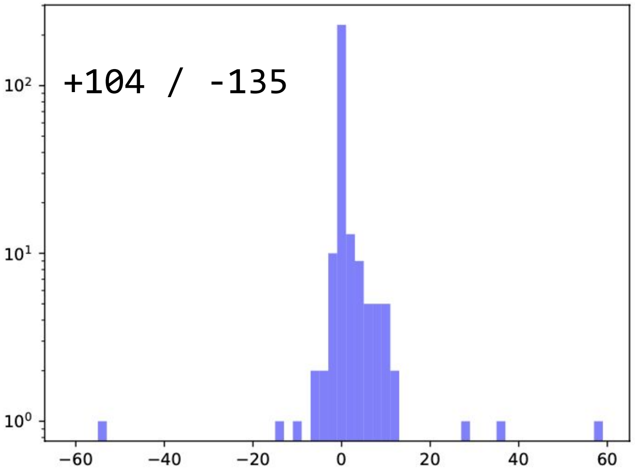
Herbie

When unsoundness is detected, Herbie has to discard the results and roll back 😭

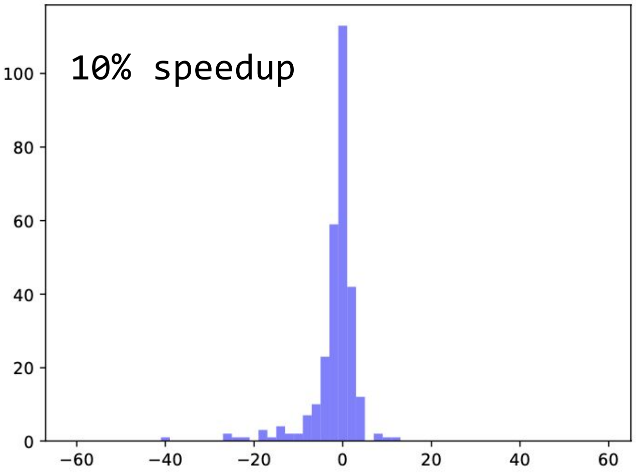
We make Herbie’s rules sound with E-graph program analyses in egglog:

- Interval analysis
- Definability analysis

Results on Herbie's benchmark



Accuracy ("# bits")



Time (s)

Results on Herbie’s benchmark

Our reimplementation in egglog achieves a comparable accuracy and performance, but does not suffer from the soundness issue in original Herbie.

Herbie’s design made simpler ✅

Bringing the power of unification to Datalog

Datalog is good at program reasoning tasks such as

- Pointer analyses.
- Type checking/inference

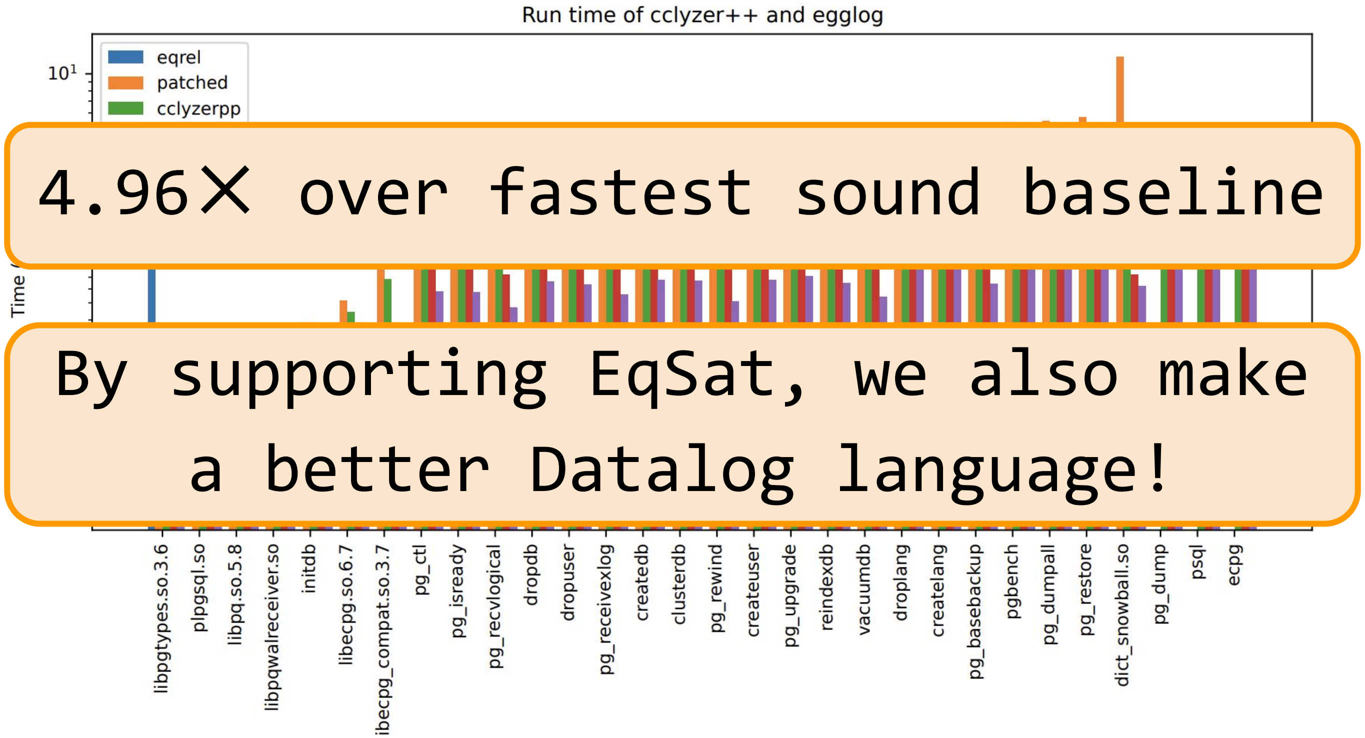
However, many advanced program reasoning tasks also require equivalence reasoning

- Steensgaard pointer analysis.
- Hindley-Milner type inference.

`Datalog:` 😟

`egglog:` ☺️

Steensgaard-style points-to analysis



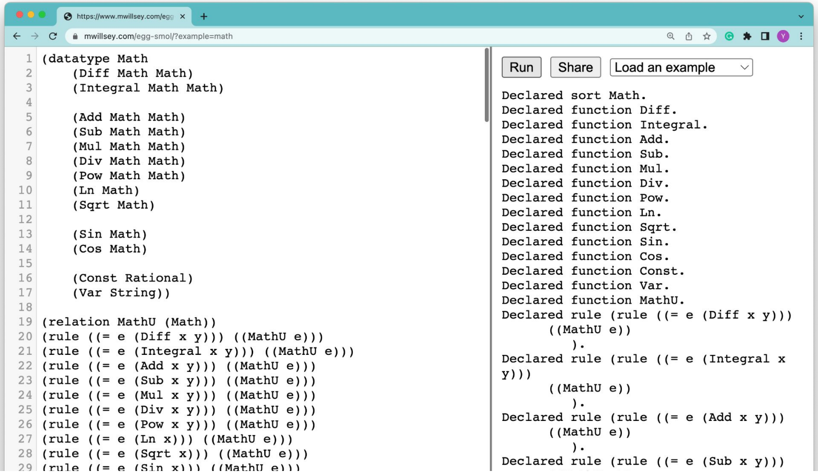
4.96× over fastest sound baseline

By supporting EqSat, we also make a better Datalog language!

egglog: Unifying Datalog and Equality Saturation

By unifying Datalog and EqSat, we get

- ✅ Fast equational reasoning *a la* EqSat.
- ✅ Rich composable analyses *a la* Datalog.
- ✅ Fast and incremental eval with DB magic.
- ✅ User-friendly language interface.


https://www.mwillsey.com/egg
mwillsey.com/egg-smol/?example=math
Run
Share
Load an example

```
(datatype Math
    (Diff Math Math)
    (Integral Math Math)

    (Add Math Math)
    (Sub Math Math)
    (Mul Math Math)
    (Div Math Math)
    (Pow Math Math)
    (Ln Math)
    (Sqrt Math)

    (Sin Math)
    (Cos Math)

    (Const Rational)
    (Var String))

(relation MathU (Math))
(rule ((= e (Diff x y))) ((MathU e)))
(rule ((= e (Integral x y))) ((MathU e)))
(rule ((= e (Add x y))) ((MathU e)))
(rule ((= e (Sub x y))) ((MathU e)))
(rule ((= e (Mul x y))) ((MathU e)))
(rule ((= e (Div x y))) ((MathU e)))
(rule ((= e (Pow x y))) ((MathU e)))
(rule ((= e (Ln x))) ((MathU e)))
(rule ((= e (Sqrt x))) ((MathU e)))
(rule ((= e (Sin x))) ((MathU e)))
```

```
Declared sort Math.
Declared function Diff.
Declared function Integral.
Declared function Add.
Declared function Sub.
Declared function Mul.
Declared function Div.
Declared function Pow.
Declared function Ln.
Declared function Sqrt.
Declared function Sin.
Declared function Cos.
Declared function Const.
Declared function Var.
Declared function MathU.
Declared rule (rule ((= e (Diff x y)))
      ((MathU e))
         ).
Declared rule (rule ((= e (Integral x
y)))
      ((MathU e))
         ).
Declared rule (rule ((= e (Add x y)))
      ((MathU e))
         ).
Declared rule (rule ((= e (Sub x y)))
```

egraphs-good.github.io/egglog

Thank you

Remy Wang

Oliver Flatt

David Cao

Philip Zucker

Eli Rosenthal

Zach Tatlock

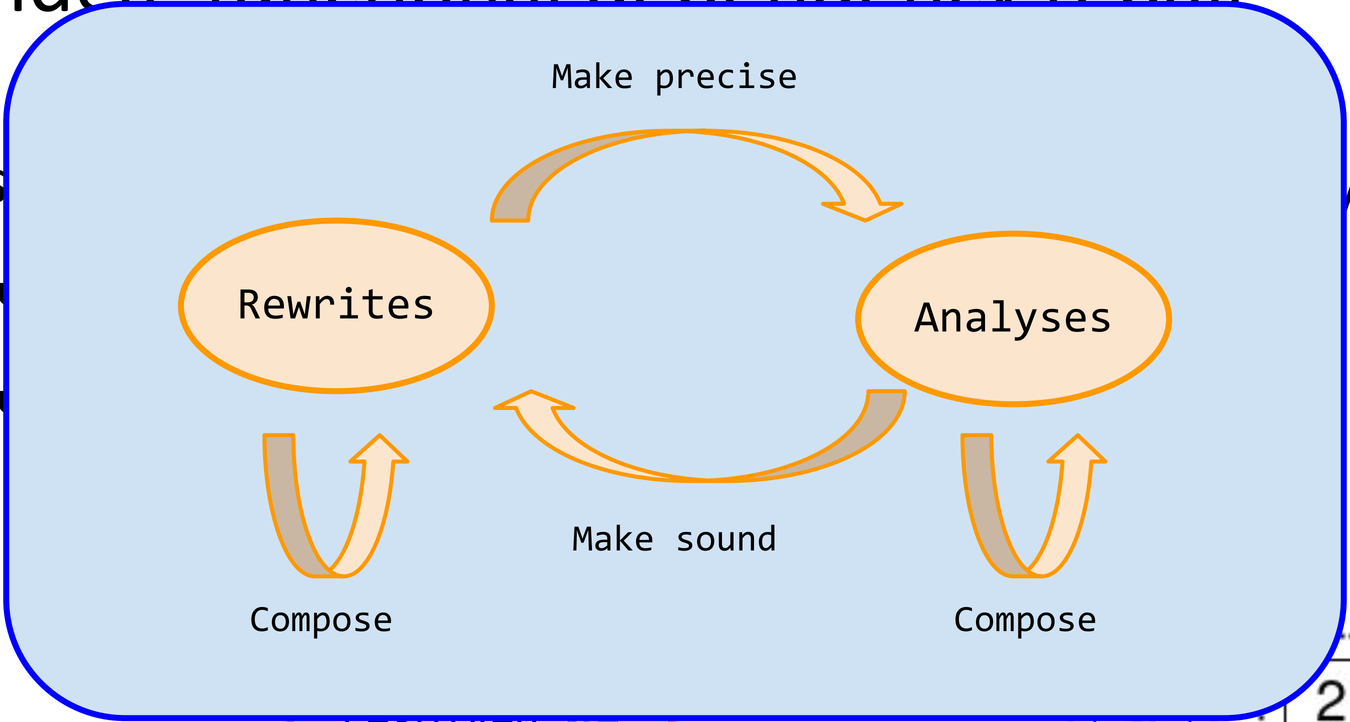
Max Willsey

Key idea: functional dependency repair

The s yses.

(fu

(fu



Make precise

Rewrites

Analyses

Make sound

Compose

Compose

resolved ✅

2