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

<table>
<thead>
<tr>
<th>In Term Rewriting with EqSat</th>
<th>In Program Analysis with Datalog</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Fast equational reasoning</td>
<td>+ Composable program analyses</td>
</tr>
<tr>
<td>- Poor analysis support</td>
<td>- Quadratic equational reasoning</td>
</tr>
</tbody>
</table>

*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]
E-graphs and Equality saturation

represents
\[(a * 2) / 2\]
E-graphs and Equality saturation

\[
x \times 2 \Rightarrow x \ll 1
\]
E-graphs and Equality saturation

x \times 2 \Rightarrow x \ll 1

(x \times y) / z \Rightarrow x \times (y / z)
E-graphs and Equality saturation

\[
x \times 2 \Rightarrow x \ll 1 \quad (x \times y) / z \Rightarrow x \times (y / z) \quad x / x \Rightarrow 1 \\
x \times 1 \Rightarrow x
\]

loop until fixpoint / timeout!
E-graphs and Equality saturation

\[ x \times 2 \Rightarrow x \ll \]

\[ x / x \Rightarrow 1 \]

\[ x \times 1 \Rightarrow x \]

\[ \text{x has to be non-zero!} \]

\[ \text{loop until fixpoint / timeout!} \]

\[ \text{Needs semantic information here} \]
E-class analyses

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

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

- Semantic analyses over E-graphs
- Each E-class is abstracted with 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!
BUT: analyses are rules, too!
// 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)
Can we do EqSat in Datalog?

We need to express in Datalog

\[(+ (+ x y) z) \rightarrow (+ x (+ y z))\]

triggers

actions
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!

\[ f(\alpha, g(\alpha)) \quad \Rightarrow \quad Q(\alpha, \text{root}) \leftarrow R_g(\alpha, x), R_f(\alpha, x, \text{root}) \]
Relational E-matching (POPL 2022)

- E-matching: pattern matching over the e-graph (triggers)
- They are just database queries! \[ f(\alpha, g(\alpha)) \quad \rightarrow \quad Q(\alpha, \text{root}) \leftarrow R_g(\alpha, x), R_f(\alpha, x, \text{root}) \]
- Significant speedups.
Relational E-matching (POPL 2022)

- E-matching: pattern matching over the e-graph (triggers)
- They are just database queries!
- Significant speedups.
- This handles triggers ✅
Relational E-matching (POPL 2022)

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

What about actions?
egglog: unifying Datalog and EqSat
egglog’s key concept: functions

Using a function-first database design

Database

relations = functions to Unit

Set of function tables

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>⇒</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>b</td>
<td>a</td>
<td>⇒</td>
</tr>
<tr>
<td>f</td>
<td>c</td>
<td>d</td>
<td>⇒</td>
</tr>
</tbody>
</table>
egglog’s key concept: functions

Using a function-first database design

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)))
```
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)))
```
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)))

<table>
<thead>
<tr>
<th>Num</th>
<th>2</th>
<th>⇒</th>
<th>C₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>C₂</td>
</tr>
<tr>
<td>Var</td>
<td>&quot;x&quot;</td>
<td>⇒</td>
<td>C₃</td>
</tr>
<tr>
<td>Add</td>
<td>C₃</td>
<td>C₁</td>
<td>⇒</td>
</tr>
<tr>
<td>Mul</td>
<td>C₂</td>
<td>C₄</td>
<td>⇒</td>
</tr>
</tbody>
</table>
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)))
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

\[(\text{datatype Math} \ (\text{Num i64}) \ \\
\quad (\text{Var String}) \ \\
\quad (\text{Add Math Math}) \ \\
\quad (\text{Mul Math Math}))\]

\[\text{;; expr} = 3 * (x + 2)\]
\[\text{(define expr} \ (\text{Mul (Num 3) (Add (Var "x") (Num 2)))))\]

\[\text{;; x + y} \Rightarrow y + x\]
\[\text{(rewrite (Add x y) (Add y x))}\]

\[\text{;; x * (y + z)} \Rightarrow x * y + x * z\]
\[\text{(rewrite (Mul x (Add y z)) (Add (Mul x y) (Mul x z)))}\]

\[\text{;; Num(x) + Num(y)} \Rightarrow \text{Num(x + y)}\]
\[\text{(rewrite (Add (Num x) (Num y)) (Num (+ x y)))}\]

\[\text{;; Num(x) * Num(y)} \Rightarrow \text{Num(x * y)}\]
\[\text{(rewrite (Mul (Num x) (Num y)) (Num (* x y)))}\]
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)))
```
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)))
```
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)))
```

**Diagram:**
- 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)))
```
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?!

\[
\begin{align*}
\text{Add} & \quad C_1 & C_3 & \Rightarrow & C_4 \\
& \quad C_2 & C_3 & \Rightarrow & C_5 \\
(\text{union} & \quad C_1 & C_2) \\

uf\text{.find}(C_2) = C_1
\end{align*}
\]
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?!

\[
\begin{array}{ccc}
\text{Add} & C_1 & C_3 \\
& C_3 & C_4 \\
\hline
\text{(union) } C_1 & C_2 & \Rightarrow & C_5
\end{array}
\]
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 $C_1$ and $C_3$ ⇒ $\{C_4, C_5\}$
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!

\[
\text{Add } C_1\quad C_3 \quad \Rightarrow \quad \{C_4, C_5\}
\]

\[
\text{uf.merge}(C_4, C_5)
\]
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!
Key idea: functional dependency repair

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

\[
\text{(function hi (Math) Rational)}
\]

\[
\text{(function lo (Math) Rational)}
\]
Key idea: functional dependency repair

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

\[
\begin{align*}
\text{(function hi (Math) Rational :merge (min old new))} \\
\text{(function lo (Math) Rational :merge (max old new))}
\end{align*}
\]
Key idea: functional dependency repair

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

\[
\begin{align*}
\text{(function } & \text{ hi (Math) Rational :merge (min old new))} \\
\text{(function } & \text{ lo (Math) Rational :merge (max old new))}
\end{align*}
\]
Key idea: functional dependency repair

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

\[
\text{(function hi (Math) Rational :merge (min old new))}
\]

\[
\text{(function lo (Math) Rational :merge (max old new))}
\]

\[
\begin{align*}
\text{hi } & \quad C_1 \quad \Rightarrow \quad 3 \\
\text{hi } & \quad C_3 \quad \Rightarrow \quad 2 \\
\text{(union } & \quad C_1, C_2) \\
\end{align*}
\]
Key idea: functional dependency repair

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

\[
\text{(function } \text{hi (Math) Rational :merge (min old new))}
\]

\[
\text{(function } \text{lo (Math) Rational :merge (max old new))}
\]
Key idea: functional dependency repair

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

\[(\text{function } hi \text{ (Math) Rational} : \text{merge (min old new)})\]

\[(\text{function } lo \text{ (Math) Rational} : \text{merge (max old new)})\]
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.

![Graph showing performance comparison between egglog and non-incremental egglog]
Herbie

\[ \sqrt{x + 1} - \sqrt{x} \quad \rightarrow \quad \frac{1}{\sqrt{1 + x} + \sqrt{x}} \]

less floating-point errors, more accurate!
Not all identities can be expressed as syntactic rules.

\[
\sqrt{x + 1} - \sqrt{x}
\]

\[
\frac{1}{\sqrt{1 + x + \sqrt{x}}}
\]

\[
\begin{align*}
x + y & \Rightarrow y + x \\
(x + y) + z & \Rightarrow x + (y + z) \\
\sin^2 x + \cos^2 x & \Rightarrow 1 \\
x^y z & \Rightarrow x^y \cdot x^z \\
x^{-1} & \Rightarrow 1 / x \\
\end{align*}
\]

\[
\begin{align*}
x \times x & \Rightarrow 1 \text{ when } x \neq 0 \\
(x \times y) / z & \Rightarrow x / (z / y) \text{ when } y \neq 0 \\
x - y & \Rightarrow (x^3 - y^3) / (x^2 + xy + y^2) \text{ when } x \neq 0 \text{ or } y \neq 0 \\
\end{align*}
\]
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)

+104 / -135

10% speedup
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.
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.

[egraphs-good.github.io/egglog](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 same mechanism also enables composable analyses.

\[
\text{function hi (Math) Rational:merge(min old new)}
\]

\[
\text{function lo (Math) Rational:merge(max old new)}
\]

[0, 3]

[2, 2]

[0, 6]

[2, 2]

hi \[ \Rightarrow \]

C\[1\]

FD violation resolved

✅

Analyses

Rewrites

Compose

Make precise

Make sound

Compose

Compose