e-graphs, four ways

Max Willsey

2023
e-graphs, four ways

- e-graphs, as data structures
- e-graphs, as datalog
- ... but fast
- ... as datalog again
\[(a \times 2) / 2\]
\[
\frac{(a \times 2)}{2} = a
\]
\[(a \times 2) / 2 \rightarrow a\]

rewrite it!

**useful**

\[
(x \times y) / z = x \times (y / z)
\]

\[
x / x = 1
\]

\[
x \times 1 = x
\]

**not so useful**

\[
x \times 2 = x << 1
\]

\[
x \times y = y \times x
\]

\[
x = x \times 1
\]
\[(a * 2) / 2 \rightarrow a * (2 / 2) \rightarrow a * 1 \rightarrow a\]

**happy path**

\[(x * y) / z = x * (y / z)\]

- \[x / x = 1\]
- \[x * 1 = x\]
\[(a \times 2) / 2 \Rightarrow (a \ll 1) / 2 \quad \text{wrong turn}\]

\[(a \times 2) / 2 \Rightarrow (2 \times a) / 2 \Rightarrow (a \times 2) / 2 \quad \text{loop}\]

\[a \Rightarrow a \times 1 \Rightarrow a \times 1 \times 1 \Rightarrow \ldots \quad \text{infinite size}\]

\[\text{pitfalls}\]

\[x \times 2 = x \ll 1\]
\[x \times y = y \times x\]
\[x = x \times 1\]

but critical for other inputs
\[(a \times 2) / 2 \rightarrow a\]

which rewrite? when?

**useful**

\[
(x \times y) / z = x \times (y / z)
\]

\[
x / x = 1
\]

\[
x \times 1 = x
\]

**not so useful**

\[
x \times 2 = x << 1
\]

\[
x \times y = y \times x
\]

\[
x = x \times 1
\]
which rewrite? when?

all of them! all the time!
\( x / x = 1 \)
\( x * 2 = x << 1 \)
\( x = x * 1 \)
\( x = l * x \)
\( e \left\langle x \right\rangle * x = z \left\langle \wedge * \right\rangle \)
\( x * y = y * x \)

**e-graphs**

+ equality saturation
e-graphs, four ways

- e-graphs, as data structures
- e-graphs, as datalog
- ... but fast
- ... as datalog again
e-graphs?

this e-class represents

\[(a \times 2) / 2\]
growing an e-graph

\( x \times 2 \rightarrow x \ll 1 \)
growing an e-graph

\[ \frac{a \times 2}{2} \quad \text{and} \quad \frac{a \ll 1}{2} \]

\[ x \times 2 \rightarrow x \ll 1 \]

this e-class represents \((a \times 2) / 2 \quad \text{and} \quad (a \ll 1) / 2\)

this e-class represents \((a \times 2) \quad \text{and} \quad (a \ll 1)\)
growing an e-graph

\[ x \times 2 \rightarrow x \ll 1 \]

\[ (x \times y) / z \rightarrow x \times (y / z) \]
e-graphs are compact

\[
\frac{a}{2} \rightarrow x \ll 1
\]

\[
\frac{x \ast y}{z} \rightarrow x \ast \frac{y}{z}
\]

\[
x \div x \rightarrow 1
\]

\[
x \ast 1 \rightarrow x
\]
saturation

✓ $x \times 2 \rightarrow x \ll 1$
✓ $(x \times y) / z \rightarrow x \times (y / z)$
✓ $x / x \rightarrow 1$
✓ $x \times 1 \rightarrow x$
this e-class represents \((a \ast 2) / 2, a, a \ast 1, \ldots\)

pick the smallest (cheapest) one

Knuth 76, Generalization of Dijkstra's Algo.
extraction

\[ x = a + \frac{1}{(y, w)} + *(x, z) \]
\[ y = *(x, w) + \langle\langle(x, z) \]
\[ z = \frac{1}{(w, w)} + 1 \]
\[ w = 2 \]

where \( f(a, b) = \min/+ \) semi-ring element in terms of \( a, b \)
equality saturation

initial term → e-graph → optimized term

do rewrites
e-matching

pattern \( f(a, g(a)) \)

subs \( \{a \mapsto 1\} \), \( f(1, g(1)) \)

subs \( \{a \mapsto 2\} \), \( f(2, g(2)) \)

e-nodes

subs \( \{a \mapsto N\} \), \( f(N, g(N)) \)

e-classes
for e-class \( c \) in e-graph \( E \):

\[
\text{for } f\text{-node } n_1 \text{ in } c:\n
\text{subst} = \{ \text{root } \mapsto c, \ \alpha \mapsto n_1.\text{child}_1 \} \n
\text{for } g\text{-node } n_2 \text{ in } n_1.\text{child}_2:\n
\text{if } \text{subst}[\alpha] = n_2.\text{child}_1:\n
\text{yield } \text{subst}\n
\]

pattern \( f(a, g(a)) \)

\( N^2 \) time, but only \( N \) matches
e-matching

• existing impls are backtracking based & complex

• doesn’t help with equality constraints

• no data complexity results
  ○ NP-hard in pattern size... e-graph size??
more than rewriting

● there's more than syntactic rewriting

● sometimes, it's useful to consider semantics
  ○ $17 + 32 \rightarrow 49$, ...

● constant folding, nullability, tensor shape, non-zero, interval arithmetic, etc, ...
more than rewriting

analyses modulo equality

• uniform interface that works in many cases
• an understanding of analyses mean
constant folding

- Option<Number> per eclass
- try to eval new e-nodes
- Option “or” on merge
constant folding

- Option<Number> per eclass
- try to eval new e-nodes
- Option “or” on merge
- it propagates up!

merge(a, 2)
e-class analysis

• 1 fact per e-class from a join-semilattice D

• make(n) → \( d_c \)
  ○ make a new analysis value for a new e-node

• join(\( d_{c_1} , d_{c_2} \)) → \( d_c \)
  ○ combine two analysis values

• modify(c) → \( c' \)
  ○ change the e-class (optionally)
constant folding

- D = Option<Number>
- make = eval
- join = option “or”
- modify = add the constant
detour: intervals

\( x \in [0, 1] \)
\( y \in [1, 2] \)

\( x + y \in [1, 3] \)
detour: intervals

\[
\begin{align*}
x \in [0, 1] & \quad 1 - \frac{2y}{x + y} \quad \text{in} \quad [-3, 1/3] \\
y \in [1, 2] & = \frac{x - y}{x + y} \quad \text{in} \quad [-2, 0] \\
& = \frac{2x}{x + y} - 1 \quad \text{in} \quad [-1, 1]
\end{align*}
\]
intervals modulo equality

\[ x \in [0, 1] \]
\[ y \in [1, 2] \]

\[
1 - \frac{2y}{x + y} \quad \text{in} \quad [-3, 1/3]
\]

\[
= \frac{x - y}{x + y} \quad \text{in} \quad [-2, 0]
\]

\[
= \frac{2x}{x + y} - 1 \quad \text{in} \quad [-1, 1]
\]

\[-1, 0\]
e-class analysis uses

- lift program analyses to e-graphs
- conditional & dynamic rewrites
  - $x / x = 1$ iff $x \neq 0$
- can express other e-graph “hacks”
  - on-the-fly extraction
\( \forall c \in G. \quad d_c = \bigvee_{n \in c} \text{make}(n) \quad \text{and} \quad \text{modify}(c) = c \)
egg: fast & easy e-graphs

- Rust library for generic e-graphs and eqsat
- packaged and documented: https://docs.rs/egg
- tutorials, industrial and academic users
e-graphs, four ways

- e-graphs, as data structures
- **e-graphs, as datalog**
- ... but fast
- ... as datalog again
why?

- equality saturation is monotonic (-ish)
  - more equalities, more terms, no "destructive" rewrites
- e-matching is the bottleneck
  - it's a backtracking search for substitutions that satisfy a formula...
schema

```
+   +   +
|   |   |
| a | b | c |
```

"+" table

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>c</td>
<td>bc</td>
</tr>
<tr>
<td>a</td>
<td>bc</td>
<td>abc</td>
</tr>
</tbody>
</table>

rewrites as rules

- example: \( x + (y + z) \rightarrow (x + y) + z \)
- \(+ (x, y, xy), +(xy, z, \text{root2}) \leftarrow +(x, yz, \text{root}), +(y, z, yz)\)
  - tempting to put root there
- not full existentials, just ADTs
  - existentials always “resolved” by FD, need a hashmap
what about the “e”? 

- e-graph is an equivalence relation
  - congruence?
- pattern matching modulo equivalence
- equivalence is **user-extensible**!
  - think EGDs from chase
eq(x, y)

- just make an equivalence relation
  - symmetric, transitive, reflexive
- all joins modulo eq
  - R(x, y), R(y, z) becomes
    R(x, \( y_1 \)), eq(\( y_1, y_2 \)), R(\( y_2 \), z)
rewrites with eq

- example: \( x + (y + z) \) \(\rightarrow\) \((x + y) + z\)
- \(+ (x, y, xy), +(xy, z, root2), eq(xy1, xy2) \leftarrow + (x, yz1, root), +(y, z, yz2), eq(yz1, yz2)\)
- non-linear patterns tend to be cyclic
  - consider \( x + (y + x) \)
congruence

- eq(z1, z2) <-
  +(x1, y1, z1), +(x2, y2, z2), eq(x1, y1), eq(x2, y2)
doesn't work

- too slow
- various tricks don't fix it
  - specialized eqrel a la Souffle,
  - subsumption
  - see Yihong Zhang's thesis
• downside of e-class analyses: there's only one
• datalog has nice, cooperating "analyses"
  ○ mutually recursive rules
• requires recursive aggregation
  ○ LowerBound(expr, number)
e-graphs, four ways

- e-graphs, as data structures
- e-graphs, as datalog
- ... but fast
- ... as datalog again
what was the problem?

• $eq(x, y)$ too slow
  ○ $n^2$ size, etc.

• no canonicalization!
if you don't canonicalize...

- e-matching only yields canonical entries!
- \( f(g(g(x))) \)
- \( f(g_1(g_3(...))), f(g_2(g_3(...))), f(g_1(g_4(...))), f(g_2(g_4(...))) \)
canonicalize

- use a union-find to define leader(x)
  - leader(x) = leader(y) iff eq(x, y)
- e-graph: explicit maintenance
  - if x = y, replace x, y, with leader(x)
  - collapse e-nodes f(x), f(y) to f(leader(x))
  - massively shrinks the e-graph
canonicalize the db

- could use some form of subsumption
  - \( f(\text{leader}(x)) := f(x) \)
  - \( f(x) \leq f(y) := x = \text{leader}(x), x \neq y \)
  - way too slow/hacky to implement in, e.g. Souffle

- let's just do what e-graphs do
  - congruence closure, "rebuilding"
no more eq

- eq relation/joins are gone!
- “semantic” equality becomes “syntactic” (again!)
- $R(x, y_1), R(y_2, z), eq(y_1, y_2)$ becomes $R(x, y), R(y, z)$
egglog

- datalog + functions + extensible equality
- examples
  - datalog: reachability
  - datalog + equality: reachability with node merging
  - eqsat: simple arithmetic optimization
egglog > eqsat

- simple implementation
  - separate optimization pass

- multipatterns
  - $a \times b = \text{split}(1, (a \oplus c) \times b)$
  - $a \times c = \text{split}(2, (a \oplus c) \times b)$

- incrementality via semi-naive
simple patterns, similar speed

above line = faster

building DB + index takes time

log scale ~ 10 million x
functions + equality

- \( f(a) = b, \ f(c) = d \)

- equality is extensible! user asserts \( a = c \)
  - what happens?
  - what happens in datalog when \( f(x, y) \) and \( f(x, z) \)
    - lattices / semirings
merge expressions

- (function foo (i64) i64 :merge (min old new))

example
- (set (foo 7) 5)
- (set (foo 7) 4)
- (foo 7) = 4
merge expressions

- also works for conflicts coming from equality
- ex: interval arithmetic
  - (function hi (Expr) Rational :merge (min old new))
  - (function lo (Math) Rational :merge (max old new))
terms?

- (function mul (Expr Expr) Expr ...)
  - :merge (union old new)
  - :default (mkset))

- congruence!

- labeled null? dynamic lattice?
the chase

- very similar to the Skolem chase
- EGDs capture FDs and "extensible" equality
- egglog has "stable" equality in a way
  - the UF prevents oscillation that can result in the chase
what's wrong?

- Built-in eq relation is really special
  - Requires infrastructure to do canonicalization
- Semantic questions
  - Why does this work? What's special about union-find?
- Limits us to equality, and only one version of it
review

good

- fast queries
- fast congruence
- functions modulo eq
  - for e.g. e-class analysis

bad

- no terms!
  - only families of terms
- semantic questions
- only supports eq
  - rebuilding, extraction are special
e-graphs, four ways

- e-graphs, as data structures
- e-graphs, as datalog
- ... but fast
- ... as datalog again

**WARNING**
braindump ahead
criteria

- no "built-in" equality
  - what are the features needed to support this?
- can’t lose terms
- fast e-matching, congruence, etc.
  - via canonicalization?
- other relations
  - partial orders, indexed equality, etc.
new encoding

- can't canonicalize the term tables
- capture the good part of the union-find
- "term tables" Add, Mul are immutable
  - a "mutable" union-find captures the eqrel
  - canonicalization rules create new terms
new encoding

- $(x + y) + z \rightarrow \text{Add}(x, y, xy), \text{Add}(xy, z, root)$
- $\text{Add}(x, y, xy_1), \text{Add}(xy_2, z, root)$
- $\text{leader}(xy_1, xy_2)$
- $\text{leader}(x, x), \text{leader}(y, y), \text{leader}(z, z)$
egglog views

- views modulo “joining relations” like eq
- \[ \text{AddEq} = \{ (\text{leader}(x), \text{leader}(y), \text{leader}(x)) \mid (x, y, z) \text{ in Add} \} \]
- some kind of notion of monotonicity
  - why does this work? what's the algebraic structure?
monotonicity?

- some kind of notion of monotonicity
  - why does this work? what's the algebraic structure?
- leader is also aggregation over eqrel
payoff

- other relations than eq
  - non-symmetric: reduction relations
  - other equalities
  - Indexed equality: eq(expr, expr) -> semiring
    - context as a set of assumptions
example

- \( a + 3b = a + b \ (mod\ 3) \)
- \( eq(x, y, \text{mod}) \)
- \( \text{plus}(a, b, \text{ab}), eq(ab, \text{root}, 3) \leftarrow \text{plus}(a, m1, \text{root}), eq(m1, m2, 3), \text{mul}(3, b, m2) \)
- \( eq(x, y, f) \leftarrow eq(x, y, k), \text{factor}(k, f) \)
payoff

- provenance
- semi-naive
  - can you re-derive congruence closure?
- top-down via demand transformation
- closer to “regular” datalog
  - other applications?
questions

• can any of this be implemented efficiently?
  ○ what algebraic structures are compatible with UF?
  ○ difficult IVM problem
  ○ queryable semirings?

• how to control the application of rules?
  ○ Demand transformation, explicit schedules, etc...

• a more flexible notion of monotonicity

• termination?