Fast and Extensible Equality Saturation

with

egg

Max Willsey, Chandrakana Nandi, Remy Wang, Oliver Flatt, Pavel Panchekha, Zachary Tatlock, and others
Fast and Extensible Equality Saturation

with egg

e-graphs good

Max Willsey, Chandrakana Nandi, Remy Wang, Oliver Flatt, Pavel Panchekha, Zachary Tatlock, and others
talk in a slide

- e-graphs and eqsat were always cool
- deferred invariant maintenance
- e-class analyses
- egg: fast & easy e-graph library
- case studies using egg

fast
extensible
our big insights
take-home message: go use it!
\[
\frac{(a \times 2)}{2}
\]

\[a\]
\((a \ast 2) / 2\) = a
\((a \times 2) / 2 \Rightarrow a\)

rewrite it!

useful
\[
\begin{align*}
(x \times y) / z &= x \times (y / z) \\
\text{ } x / x &= 1 \\
\text{ } x \times 1 &= x
\end{align*}
\]

not so useful
\[
\begin{align*}
x \times 2 &= x \ll 1 \\
x \times y &= y \times x \\
x &= x \times 1
\end{align*}
\]
\[(a \times 2) / 2 \Rightarrow a \times (2 / 2) \Rightarrow a \times 1 \Rightarrow a\]

**happy path**

\[
\frac{x \times y}{z} = x \times \frac{y}{z}
\]

\[
x / x = 1
\]

\[
x \times 1 = x
\]
(a * 2) / 2 → (a << 1) / 2  x * y = y * x  

(a * 2) / 2 → (2 * a) / 2 → (a * 2) / 2  loop

a → a * 1 → a * 1 * 1 → infinite size

wrong turn

but critical for other inputs

pitfalls

(a * 2) / 2 → (a * 2) / 2

x * 2 = x << 1
x * y = y * x
x = x * 1

critical

other inputs

infinite size

loop
\[(a \ast 2) / 2 \Rightarrow a\]

which rewrite? when?

\[
\begin{align*}
\text{useful} & \quad (x \ast y) / z = x \ast (y / z) \\
x / x = 1 \\
x \ast 1 = x
\end{align*}
\]

\[
\begin{align*}
\text{not so useful} & \quad x \ast 2 = x \ll 1 \\
x \ast y = y \ast x \\
x = x \ast 1
\end{align*}
\]
which rewrite? when?  
all of them! all the time!
which rewrite? when?

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

\[x / x = 1\]

\[x \times 1 = x\]

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

\[x = x \times 1\]

all of them! all the time!

equality saturation!

faster!

more flexible!
e-graphs?

this e-class represents

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

this e-class represents $(a * 2) / 2$ and $(a << 1) / 2$

this e-class represents $(a * 2)$ and $(a << 1)$

$x * 2 \rightarrow x << 1$
growing an e-graph

\[
\frac{x \times 2}{a} \rightarrow x \ll 1 \\
\frac{(x \times y)}{z} \rightarrow x \times \frac{y}{z} 
\]
e-graphs are compact

\[
\begin{align*}
\text{a, a * 1, a * 1 * 1, …}
\end{align*}
\]
saturation

✓ $x \times 2 \rightarrow x \ll 1$
✓ $(x \times y) \div z \rightarrow x \times (y \div z)$
✓ $x \div x \rightarrow 1$
✓ $x \times 1 \rightarrow x$
equality saturation

initial term → e-graph

extract

optimized term

do rewrites
equality saturation

initial term $\rightarrow$ e-graph $\rightarrow$ optimized term

congruence

if $a = b$ then $f(a) = f(b)$

restore invariants

find a pattern

apply a match

loop runs many times!
talk in a slide

- e-graphs and eqsat were always cool
- deferred invariant maintenance
- e-class analyses
- egg: fast & easy e-graph library
- case studies in using egg

our big insights

take-home message: go use it!
def equality_saturation(expr, rewrites):
    egraph = initial_egraph(expr)

    while not egraph.is_saturated_or_timeout():
        for rw in rewrites:
            for (subst, ec) in egraph.ematch(rw.lhs):
                ec2 = egraph.add(rw.rhs.subst(subst))
                egraph.merge(ec, ec2)

    return egraph.extract_best()
equality saturation

```python
def equality_saturation(expr, rewrites):
    egraph = initial_egraph(expr)

    while not egraph.is_saturated_or_timeout():
        for rw in rewrites:
            for (subst, ec) in egraph.ematch(rw.lhs):
                ec2 = egraph.add(rw.rhs.subst(subst))
                egraph.merge(ec, ec2)

    return egraph.extract_best()
```

- rewrites are ordered
- read/write interleaved
  - more invariant maintenance
- invariants baked-in
def equality_saturation(expr, rewrites):
    egraph = initial_egraph(expr)
    while not egraph.is_saturated_or_timeout():
        matches = []
        for rw in rewrites:
            for (subst, ec) in egraph.ematch(rw.lhs):
                ec2 = egraph.add(rw.rhs.subst(subst))
                egraph.merge(ec, ec2)
        for (rw, subst, ec) in matches:
            egraph.rebuild()
    return egraph.extract_best()
deferred invariant maintenance

initial term -> e-graph -> optimized term

- restore all invariants
- find all patterns
- apply all matches
rebuilding is faster
rebuilding is faster
why is rebuilding faster?

- consider $f_1(x) \ldots f_n(x)$ and $y_1 \ldots y_n$
- workload: merge($x, y_1$) $\ldots$ merge($x, y_n$)
- traditional: $O(n^2)$ hashcons updates
- deferred only does $O(n)$

Downey, Sethi, Tarjan 1980
talk in a slide

- e-graphs and eqsat were always cool
- deferred invariant maintenance
- e-class analyses
- egg: fast & easy e-graph library
- case studies in using egg

our big insights
take-home message: go use it!
rewriting is great, but...

- there's more than *syntactic* rewriting
- sometimes, it's useful to consider *semantics*
- constant folding, nullability, tensor shape, non-zero, interval arithmetic, etc, ...
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!
what now?

- values are “stuck” in the analysis
- modify hook
- add (or prune) e-nodes
constant folding

- not syntactic
- consider $a = 2$
- we know $a \times 2 = 2 \times 2 = 4$
- facts about children can entail facts about parents
e-class analysis

- 1 fact per e-class from a join-semilattice $D$
- $\text{make}(n) \rightarrow d_c$
  - make a new analysis value for a new e-node
- $\text{join}(d_{c1}, d_{c2}) \rightarrow d_c$
  - combine two analysis values
- $\text{modify}(c) \rightarrow c'$
  - change the e-class (optionally)
constant folding

- \( D = \text{Option<Number>} \)
- \( \text{make} = \text{eval} \)
- \( \text{join} = \text{option “or”} \)
- \( \text{modify} = \text{add the constant} \)
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”
  - debugging, pruning, on-the-fly extraction
e-class analysis invariant

\[ \forall c \in G. \quad d_c = \bigwedge_{n \in c} \text{make}(n) \quad \text{and} \quad \text{modify}(c) = c \]

Analysis data is LUB (lattice properties)
talk in a slide

- e-graphs and eqsat were always cool
- deferred invariant maintenance
- e-class analyses

egg: fast & easy e-graph library
- case studies in using egg

our big insights

take-home message: go use it!
egg: fast & easy e-graphs

- Rust library for generic e-graphs and eqsat
- packaged and documented: [https://docs.rs/egg](https://docs.rs/egg)
- easy to use, people are building on it
**egg features**

- pre-made, customizable eqsat “outer loop”
  - logging, rule scheduling, saturation checking
- custom rewrites that run arbitrary code
- batch simplification
egg case studies

- Herbie: floating point
  - 3000x faster
- SPORES: linear algebra kernels
  - 1.2-5x better
- ML compute graphs
  - 23% better, 48x faster
- Szalinski: CAD synthesis
  - 12,000 part eval < 1s synthesis
- ... TVM, Java testing, vectorization, hw/sw co-design, educational problems, ...
egg: e-graphs good

egraphs-good.github.io

- e-graphs and eqsat were always cool
- deferred invariant maintenance
- e-class analyses
- egg: fast & easy e-graph library
- case studies in using egg

our big insights

take-home message: go use it!