\[(a \times 2) / 2 \Rightarrow a\]
(a * 2) / 2 ⇒ a

REWRITE!
\[(a \times 2) / 2 \Rightarrow a\]

**REWRITE!**

**Useful**

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

\[x / x = 1\]

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

**REWRITE!**

**Useful**

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

\[x / x = 1\]

\[x \times 1 = x\]

**Less Useful**

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

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

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

“happy path”

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

\[ x / x = 1 \]

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

"happy path"

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

\[x / x = 1\]

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

“happy path”

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

\[x / x = 1\]

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

“happy path”
\[(x \times y) / z = x \times (y / z)\]
\[x / x = 1\]
\[x \times 1 = x\]
\[(a \times 2) / 2 \Rightarrow a \times (2 / 2) \Rightarrow a \times 1 \Rightarrow a\]

“happy path”

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

\[x / x = 1\]

\[x \times 1 = x\]
Pitfalls

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

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

\[ x \ast y = y \ast x \]

\[ x = x \ast 1 \]
\[(a \times 2) / 2 \Rightarrow (a \ll 1) / 2\]

**Pitfalls**

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

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

\[x = x \times 1\]
\[(a \times 2) / 2 \Rightarrow (a \ll 1) / 2 \]  ❌ order

**Pitfalls**

- \[x \times 2 = x \ll 1\]
- \[x \times y = y \times x\]
- \[x = x \times 1\]
(a * 2) / 2 ⇒ (a << 1) / 2  ❌ order

(a * 2) / 2

Pitfalls

x * 2 = x << 1
x * y = y * x
x = x * 1
\[(a \ast 2) / 2 \implies (a \ll 1) / 2 \]
\[(a \ast 2) / 2 \implies (2 \ast a) / 2 \]

**Pitfalls**
- \(x \ast 2 = x \ll 1\)
- \(x \ast y = y \ast x\)
- \(x = x \ast 1\)
\[(a \times 2) / 2 \Rightarrow (a \ll 1) / 2 \quad \times \text{order} \]

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

Pitfalls

\[
x \times 2 = x \ll 1 \\
x \times y = y \times x \\
x = x \times 1
\]
(a * 2) / 2 \Rightarrow (2 * a) / 2 \Rightarrow (a * 2) / 2 \ \times \ \text{order diverge}

\text{Pitfalls}
\begin{align*}
x \ast 2 &= x \ll 1 \\
x \ast y &= y \ast x \\
x &= x \ast 1
\end{align*}
\begin{align*}
(a \times 2) / 2 & \Rightarrow (a \ll 1) / 2 \quad \text{x order} \\
(a \times 2) / 2 & \Rightarrow (2 \times a) / 2 \Rightarrow (a \times 2) / 2 \quad \text{x diverge}
\end{align*}

\textbf{Pitfalls}

\begin{align*}
x \times 2 &= x \ll 1 \\
x \times y &= y \times x \\
x &= x \times 1
\end{align*}
(a * 2) / 2 \Rightarrow (a << 1) / 2 \quad \text{❌ order}

(a * 2) / 2 \Rightarrow (2 * a) / 2 \Rightarrow (a * 2) / 2 \quad \text{❌ diverge}

a \Rightarrow a * 1

\textbf{Pitfalls}

\begin{align*}
  x * 2 &= x << 1 \\
  x * y &= y * x \\
  x &= x * 1
\end{align*}
(a * 2) / 2 ⇒ (a << 1) / 2

(a * 2) / 2 ⇒ (2 * a) / 2 ⇒ (a * 2) / 2

diverge

a ⇒ a * 1 ⇒ a * 1 * 1

Pitfalls

x * 2 = x << 1
x * y = y * x
x = x * 1
\[(a \times 2) / 2 \Rightarrow (a \ll 1) / 2 \quad \text{x order}\]

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

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

**Pitfalls**

- \[x \times 2 = x \ll 1\]
- \[x \times y = y \times x\]
- \[x = x \times 1\]
(a * 2) / 2 ⇒ (a << 1) / 2  ❌ order
(a * 2) / 2 ⇒ (2 * a) / 2 ⇒ (a * 2) / 2  ❌ converge
da ⇒ a * 1 ⇒ a * 1 * 1 ⇒ …  ❌ infinite size

Critical for other inputs!

Pitfalls

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

Which rewrite? When?

**Useful**
\[
(x \times y) / z = x \times (y / z)
\]
\[
x / x = 1
\]
\[
x \times 1 = x
\]

**Less Useful**
\[
x \times 2 = x \ll 1
\]
\[
x \times y = y \times x
\]
\[
x = x \times 1
\]
(a * 2) / 2 ⇒ a

Which rewrite? When?

Equality Saturation

Try applying all the rules in every order!
\[(a \times 2) / 2 \Rightarrow a\]

Which rewrite? When?

Equality Saturation

Try applying all the rules in every order?!
E-graphs

- Used for congruence closure (Downey, Sethi, Tarjan 1980)
  - Intuition: union-find (Tarjan 1975) but function-aware
- Key for equality and uninterpreted funcs (EUF) theory in SMT
  - Intuition: the “glue” that connects other theories to SAT
- Historically: “baked in” to SMT solvers, no general libraries 😞
E-graphs
E-graphs

E-classes contain e-nodes (ops)
E-graphs

e-classes contain e-nodes (ops)

e-nodes’ arguments are e-classes!
E-graphs

- E-classes contain e-nodes (ops)
- E-nodes’ arguments are e-classes!
- E-graphs maximize sharing (no copies of same e-node)
E-graphs

This e-classes represents

\((a \times 2) / 2\)
E-graphs: applying rewrite rules

\[ x \times 2 \rightarrow x \ll 1 \]
E-graphs: applying rewrite rules

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

This e-classes represents \( a * 2 \) and \( a \ll 1 \)
E-graphs: applying rewrite rules

This e-classes represents \((a*2)/2\) and \((a<<1)/2\)

This e-classes represents \(a*2\) and \(a<<1\)

\(x * 2 \rightarrow x << 1\)
E-graphs: applying rewrite rules

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

This e-classes represents a*2 and a<<1

E-graphs never forget. Rewrites don’t lose info!
E-graphs: applying rewrite rules

\[
\begin{align*}
x \times 2 &\rightarrow x \ll 1 \\
(x \times y)/z &\rightarrow x \times (y/z)
\end{align*}
\]
E-graphs: applying rewrite rules

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

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

\[ x / x \rightarrow 1 \]

\[ x \times 1 \rightarrow x \]
E-graphs: compact representation

Rewrites can **shrink** e-graphs!
- $6 \rightarrow 5$ eclasses

E-graphs can represent $\infty$ terms
- $a, a \ast 1, a \ast 1 \ast 1, \ldots$

E-graphs can “**saturate**”
- learn all derivable eqs ✅

$x / x \rightarrow 1$
$x \ast 1 \rightarrow x$
Equality Saturation

● Technique first used in Denali (Joshi, Nelson, Randall 2002)
  ○ Optimizing straight-line assembly kernels for Alpha

● Extended to loops in Peggy [POPL 2009]
  ○ Coined term “Equality Saturation”
  ○ Coinductive stream operators for algebraic loop rewrites
  ○ Used Rete algo from expert sys for incremental e-matching
Equality Saturation

initial term
Equality Saturation

initial term $\Rightarrow$ e-graph
Equality Saturation

initial term $\rightarrow$ e-graph

rewrite
Equality Saturation

initial term → e-graph

rewrite

till saturation or timeout
Equality Saturation

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

extract

rewrite $\Rightarrow$

till saturation or timeout
Equality Saturation

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

- greedy (size), ILP (CSE), GA, …
- extract
- rewrite
- till saturation or timeout
Equality Saturation

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

rewrite

[Diagram showing the process of Equality Saturation with initial term transforming into an e-graph, which then optimizes to an optimized term through rewrite.]
Equality Saturation

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

find pattern ("e-match")
Equality Saturation

Initial term $\rightarrow$ e-graph $\rightarrow$ Optimized term

- Find pattern ("e-match")
- Apply match
Equality Saturation

1. Initial term → e-graph
2. Find pattern ("e-match")
3. Apply match
4. Restore invariants
5. Optimized term
Equality Saturation

- **Initial Term** → **E-graph** → **Optimized Term**

  - *Restore Invariants*
  - *Find Pattern* ("e-match")
  - *Apply Match*

  **Congruence**
  \[ a = b \implies f(a) = f(b) \]
Equality Saturation

- initial term \(\rightarrow\) e-graph \(\rightarrow\) optimized term

- restore invariants
- find pattern ("e-match")
- apply match

**congruence**

\[a = b \Rightarrow f(a) = f(b)\]
Equality Saturation

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

Goal: make it fast!

Restore invariants

hot loop ("e-match")

find pattern

apply match

congruence

$a = b \Rightarrow f(a) = f(b)$
egg EqSat Toolkit [POPL 2021, Distinguished Paper]

- Deferred invariant maintenance & batching
- Relational e-matching [POPL 2022]
- E-class analyses
- Rewrite rule synthesis with Ruler [OOPSLA 2021, Distinguished Paper]

Applications

- 3D CAD in Szalinski, FP Accuracy in Herbie, Lib Learning in Babble, …
- EVM simplify @ Certora, wasm JIT @ Fastly, datapath optimize @ Intel, …
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()
```
Equality Saturation

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

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

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 maint
- invariants baked-in
Deferred Invariant Maintenance in egg

```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()
```

```python
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):
                matches.append(((rw, subst, ec))
                ec2 = egraph.add(rw.rhs.subst(subst))
                egraph.merge(ec, ec2)

    return egraph.extract_best()
```
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:
            ec2 = egraph.add(rw.rhs.subst(subst))
            egraph.merge(ec, ec2)
    egraph.rebuild()

    return egraph.extract_best()
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)
    egraph.rebuild()
    return egraph.extract_best()
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)
    egraph.rebuild()
    return egraph.extract_best()
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):
                matches.append((rw, subst, ec))
        for (rw, subst, ec) in matches:
            ec2 = egraph.add(rw.rhs.subst(subst))
            egraph.merge(ec, ec2)
    egraph.rebuild()
    return egraph.extract_best()
versus
Rebuilding is faster.
Rebuilding is faster

Test is 30x faster
Why is rebuilding is faster?

- Consider $f_1(x) \ldots f_n(x)$ and $y_1 \ldots y_n$
- Workload: $\text{merge}(x, y_1) \ldots \text{merge}(x, y_n)$
- Traditional: $O(n^2)$ hashcons updates
- Deferred only does $O(n)$ updates

Downey, Sethi, Tarjan 1980
Why is rebuilding is faster?
Why is rebuilding is faster?
Why is rebuilding faster?
Why is rebuilding is faster?
More amortization via *batching* in egg

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

- restore *all* invariants
- find *all* patterns
- apply *all* matches
More amortization via *batching* in egg

- Initial terms
- E-graph
- Optimized term

- Chunk entire set of inputs into a single e-graph!
- Restore all invariants
- Find all patterns
- Apply all matches
More amortization via *batching* in egg

- **Initial terms**
  - Chunk entire set of inputs into a single e-graph!

- **E-graph**
  - Restore all invariants
  - Apply all matches

- **Optimized term**
  - Find all patterns
  - Extract optimized term from each root.
More amortization via *batching* in egg

1. **Initial terms**
2. **E-graph**
3. **Optimized term**

- **Chunk entire set of inputs into a single e-graph!**
- **Shared optimization** + “e-graph seeding”
- **Extract optimized term from each root.**
E-graphs in Herbie

\[
\frac{(-b) + \sqrt{b \cdot b - 4 \cdot (a \cdot c)}}{2 \cdot a}
\]
E-graphs in Herbie

\[ \frac{(-b) + \sqrt{b \cdot b - 4 \cdot (a \cdot c)}}{2 \cdot a} \]
E-graphs in Herbie

\[
\frac{(-b) + \sqrt{b \cdot b - 4 \cdot (a \cdot c)}}{2 \cdot a}
\]

\[
\downarrow
\]

**if** \( b \leq -2.1714197031320663 \cdot 10^{+114} \):

\[-\frac{b}{a}\]

**elif** \( b \leq 2.9809086538561536 \cdot 10^{-153} \):

\[
\frac{\sqrt{\text{fma}(b, b, c \cdot (a-4))} - b}{a \cdot 2}
\]

**elif** \( b \leq 3.095118518558678 \cdot 10^{+20} \):

\[t_0 := 4 \cdot (a \cdot c)\]

\[
\frac{t_0 \cdot 0.5}{(-b) - \sqrt{b \cdot b - t_0}}
\]

**else**:

\[-\frac{c}{b}\]
E-graphs in Herbie

\[
\frac{(-b) + \sqrt{b \cdot b - 4 \cdot (a \cdot c)}}{2 \cdot a}
\]

\[
\downarrow
\]

**if** \( b \leq -2.1714197031320663 \cdot 10^{+114} : \)

\[-\frac{b}{a} \]

**elif** \( b \leq 2.9809086538561536 \cdot 10^{-153} : \)

\[
\frac{\sqrt{\text{fma}(b, b, c \cdot (a - 4)) - b}}{a^2}
\]

**elif** \( b \leq 3.095118518558678 \cdot 10^{+20} : \)

\[
t_0 := 4 \cdot (a \cdot c)
\]

\[
t_0 \cdot \frac{0.5}{a}
\]

\[
(-b) - \sqrt{b \cdot b - t_0}
\]

**else :**

\[-\frac{c}{b} \]
\((-b) + \sqrt{b^2 - 4ac}\)
egg EqSat Toolkit [POPL 2021, Distinguished Paper]

✓ Deferred invariant maintenance & batching
❑ Relational e-matching [POPL 2022]
❑ E-class analyses
❑ Rewrite rule synthesis with Ruler [OOPSLA 2021, Distinguished Paper]
❑ Applications
  ❑ 3D CAD in Szalinski, FP Accuracy in Herbie, Lib Learning in Babble, …
  ❑ EVM simplify @ Certora, wasm JIT @ Fastly, datapath optimize @ Intel, …
egg’s Equality Saturation

- initial term
- e-graph
- optimized term

- restore all invariants
- find all patterns
- apply all matches
egg’s Equality Saturation

- initial term
- e-graph
- optimized term

1. Find all patterns
2. Apply all matches
3. Restore all invariants

Now that this is fast…
egg's Equality Saturation

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

- restore all invariants
- find all patterns

Now that this is fast... we bottleneck on matching 🙁
E-matching: pattern matching over e-graphs

- *E-matching*: find subssts from pattern variables to e-classes
- Substs guaranteed to be represented by the matched e-graph
E-matching: pattern matching over e-graphs

- *E-matching*: find substs from pattern variables to e-classes
- Substs guaranteed to be represented by the matched e-graph
E-matching: pattern matching over e-graphs

- **E-matching**: find substs from pattern variables to e-classes
- Substs guaranteed to be represented by the matched e-graph

\[ f(\alpha, g(\alpha)) \text{ will match } f(1, g(1)) \quad f(2, g(2)) \quad \ldots \quad f(N, g(N)) \]

\[ \{ \alpha \mapsto 1 \} \quad \{ \alpha \mapsto 2 \} \quad \ldots \quad \{ \alpha \mapsto N \} \]
E-matching: pattern matching over e-graphs

- **E-matching**: find subssts from pattern variables to e-classes
- Substs guaranteed to be represented by the matched e-graph

\[
\begin{align*}
\text{\texttt{f}(\alpha, \texttt{g}(\alpha))} & \text{ will match } \\
\text{\texttt{f}(1, \alpha)} & \text{ will match }
\end{align*}
\]

\[
\begin{align*}
\text{\texttt{f}(1, \texttt{g}(1))} & \quad \{\alpha \mapsto 1\} \\
\text{\texttt{f}(2, \texttt{g}(2))} & \quad \{\alpha \mapsto 2\} \\
\text{\texttt{f}(N, \texttt{g}(N))} & \quad \{\alpha \mapsto N\}
\end{align*}
\]

\[
\begin{align*}
\text{\texttt{f}(1, \texttt{g}(1))} & \quad \{\alpha \mapsto c_g\}. \\
\text{\texttt{f}(1, \texttt{g}(2))}, \text{ witnessed by } & \quad \{\alpha \mapsto c_g\}. \\
\text{\texttt{f}(1, \texttt{g}(N))} & \quad \text{...}
\end{align*}
\]
E-matching: pattern matching over e-graphs

- **E-matching**: find subsits from pattern variables to e-classes
- Substs guaranteed to be represented by the matched e-graph
- NP-complete wrt to pattern size (Kozen 1977)

\[ f(\alpha, g(\alpha)) \] will match \[ f(1, g(1)), f(2, g(2)), \ldots, f(N, g(N)) \], witnessed by \[ \{ \alpha \mapsto 1 \}, \{ \alpha \mapsto 2 \}, \ldots, \{ \alpha \mapsto N \} \].

\[ f(1, \alpha) \] will match \[ f(1, g(1)), f(1, g(2)), \ldots, f(1, g(N)) \], witnessed by \[ \{ \alpha \mapsto c_g \}, \{ \alpha \mapsto c_g \}, \ldots \].
E-matching: pattern matching over e-graphs

- **E-matching**: find subst from pattern variables to e-classes
- Substs guaranteed to be represented by the matched e-graph
- NP-complete wrt to pattern size (Kozen 1977)

\[
f(\alpha, g(\alpha)) \text{ will match } f(1, g(1)) f(2, g(2)) \ldots f(N, g(N)), \text{ witnessed by } \{\alpha \mapsto 1\} \{\alpha \mapsto 2\} \ldots \{\alpha \mapsto N\}.
\]

\[
f(1, \alpha) \text{ will match } f(1, g(1)) f(1, g(2)) \ldots f(1, g(N)), \text{ witnessed by } \{\alpha \mapsto c_g\}.
\]

But patterns are often small...
E-matching: pattern matching over e-graphs

- **E-matching**: find substs from pattern variables to e-classes
- Substs guaranteed to be represented by the matched e-graph
- NP-complete wrt to pattern size (Kozen 1977)

\[ f(\alpha, g(\alpha)) \]  
will match  
\[ f(1, g(1)), f(2, g(2)), \ldots, f(N, g(N)) \]
witnessed by  
\[ \{ \alpha \mapsto 1 \}, \ldots, \{ \alpha \mapsto N \} \]

\[ f(1, \alpha) \]  
will match  
\[ f(1, g(1)), f(1, g(2)), \ldots, f(1, g(N)) \]
witnessed by  
\[ \{ \alpha \mapsto c \}, \{ \alpha \mapsto g \} \].

But patterns are often small…

💡 # of matches is much better metric!
Traditional e-matching via backtracking
Traditional e-matching via backtracking

\[
f(\alpha, g(\alpha))
\]

for e-class c in e-graph E:
for e-class $c$ in e-graph $E$:

for $f$-node $n_1$ in $c$:

$f(\alpha, g(\alpha))$

Backtracking search $f(\alpha, g(\alpha))$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$
Traditional e-matching via backtracking

\[ f(\alpha, g(\alpha)) \]

for e-class \( c \) in e-graph \( E \):
for f-node \( n_1 \) in \( c \):
    \[ \text{subst} = \{ \text{root} \mapsto c, \ \alpha \mapsto n_1.\text{child}_1 \} \]

Backtracking search \( f(\alpha, g(\alpha)) \)
for e-class \( c \) in e-graph \( E \):
for f-node \( n_1 \) in \( c \):
    subst = \{root \mapsto c, \alpha \mapsto n_1.\text{child}_1\}
for g-node \( n_2 \) in \( n_1.\text{child}_2 \):
    f(\alpha, g(\alpha))

Backtracking search \textit{f(a, g(a))}
Traditional e-matching via backtracking

for e-class c in e-graph E:
  for f-node n₁ in c:
    subst = {root ↦ c, α ↦ n₁.child₁}
    for g-node n₂ in n₁.child₂:
      if subst[α] = n₂.child₁:
        f(α, g(α))

Backtracking search f(a, g(α))

f(1,c₁) g(1) ✓
g(2) □
...
g(N) □
f(2,c₂) g(1) □
g(2) ✓
...
g(N) □
...
f(N,cₙ) g(1) □
g(2) □
...
g(N) ✓
Traditional e-matching via backtracking via backtracking

\[
f(\alpha, g(\alpha))
\]

for e-class \( c \) in e-graph \( E \):
  for f-node \( n_1 \) in \( c \):
    \[
    \text{subst} = \{ \text{root} \mapsto c, \ \alpha \mapsto n_1.\text{child}_1 \}
    \]
    for g-node \( n_2 \) in \( n_1.\text{child}_2 \):
      if \( \text{subst}[\alpha] = n_2.\text{child}_1 \):
        yield \( \text{subst} \)
Traditional e-matching via backtracking

\[ f(\alpha, g(\alpha)) \]

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

for f-node \( n_1 \) in \( c \):

\[ \text{subst} = \{ \text{root} \mapsto c, \alpha \mapsto n_1.\text{child}_1 \} \]

for g-node \( n_2 \) in \( n_1.\text{child}_2 \):

if \( \text{subst}[\alpha] = n_2.\text{child}_1 \):

yield \( \text{subst} \)

O\((N^2)\), yet at most O\((N)\) matches

Traditional e-matching via backtracking

\[ f(\alpha, g(\alpha)) \]

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

for \( f\)-node \( n_1 \) in \( c \):

\[ \text{subst} = \{ \text{root} \mapsto c, \alpha \mapsto n_1.\text{child}_1 \} \]

for \( g\)-node \( n_2 \) in \( n_1.\text{child}_2 \):

if \( \text{subst}[\alpha] = n_2.\text{child}_1 \):

yield \( \text{subst} \)

O\((N^2)\), yet at most O\((N)\) matches
Many optimizations in literature
  ○ custom VMs for “CSE”
  ○ specific patterns
  ○ mod-time analysis

No data complexity bounds!
💡 Key insight: e-matching is a DB problem!

E-matching in e-graphs
Finding substitutions such that substituted terms are represented in an e-graph.

Conjunctive queries in DBs
Finding substitutions such that substituted atoms are present in a relational DB.
egg’s relational e-matching
egg’s relational e-matching

- Given e-graph + patterns
egg’s relational e-matching

- Given e-graph + patterns
- Transform e-graph to tables
egg’s relational e-matching

- Given e-graph + patterns
- Transform e-graph to tables
- Compile patterns to queries

Given e-graph + patterns
Transform e-graph to tables
Compile patterns to queries
egg’s relational e-matching

- Given e-graph + patterns
- Transform e-graph to tables
- Compile patterns to queries
- Use DB query engine to e-match!
egg’s relational e-matching

- Given e-graph + patterns
- Transform e-graph to tables
- Compile patterns to queries
- Use DB query engine to e-match!
- Derive bounds from DB theory!

\[
f(\alpha, g(\alpha))
\]
\[
g(f(\alpha, \alpha))
\]

[Diagram showing e-graph transformation to tables and queries]

\[
Q(\text{root}, \alpha) \leftarrow R_f(\text{root}, \alpha, x), R_g(x, \alpha)
\]
\[
Q(\text{root}, \alpha) \leftarrow R_g(\text{root}, x), R_f(x, \alpha, \alpha)
\]

[Database with green check mark]
E-graphs as tables (relational DBs)

\[ \begin{array}{ccc}
\text{id} & \text{arg}_1 & \text{arg}_2 \\
\hline
\text{id} & \text{arg}_1 & \text{arg}_2 \\
\hline
\text{id} & \text{arg}_1 & \text{arg}_2
\end{array} \]

\begin{align*}
R_f & : \\
\begin{array}{ccc}
\text{id} & \text{arg}_1 & \text{arg}_2 \\
\hline
\text{id} & \text{arg}_1 & \text{arg}_2
\end{array} \\
R_g & : \\
\begin{array}{ccc}
\text{id} & \text{arg}_1 & \text{arg}_2 \\
\hline
\text{id} & \text{arg}_1 & \text{arg}_2
\end{array} \\
R_i=1\ldots N & : \\
\begin{array}{ccc}
\text{id} & \text{arg}_1 & \text{arg}_2 \\
\hline
\text{id} & \text{arg}_1 & \text{arg}_2
\end{array} \\
\end{align*}
E-graphs as tables (relational DBs)

Every e-node becomes a row
E-match patterns as conjunctive queries

\[ f(\alpha, g(\alpha)) \]
E-match patterns as conjunctive queries

\[ f(\alpha, g(\alpha)) \]

\[ Q(\text{root, } \alpha) \leftarrow R_f(\text{root, } \alpha, x), R_g(x, \alpha) \]
E-match patterns as conjunctive queries

\[ f(\alpha, g(\alpha)) \]

\[ Q(root, \alpha) \leftarrow R_f(root, \alpha, x), R_g(x, \alpha) \]

\[ \text{ind} = \{\} \]

\[ \text{for } (x, \alpha) \text{ in } R_g : \text{ # build index} \]
\[ \text{ind}.\text{insert}((x, \alpha)) \]

\[ R_g(c_g, 1) \]
\[ R_g(c_g, 2) \]
\[ \ldots \]
\[ R_g(c_g, N) \]

build hash
E-match patterns as conjunctive queries

\[ f(\alpha, g(\alpha)) \]

\[ Q(\text{root}, \alpha) \leftarrow R_f(\text{root}, \alpha, x), R_g(x, \alpha) \]

\[
\text{ind} = \{\}
\]
\[
\text{for } (x, \alpha) \text{ in } R_g: \quad \# \text{ build index}
\]
\[
\text{ind}.\text{insert}((x, \alpha))
\]
\[
\text{for } (\text{root}, \alpha, x) \text{ in } R_f: \quad \# \text{ probe}
\]
\[
\text{if } (\alpha, x) \text{ in ind:}
\]
\[
\text{yield } \{\text{root }\mapsto \text{root}, \alpha \mapsto \alpha\}
\]

\[
\begin{align*}
\text{build hash} & \quad R_g(c_g, 1) \quad \checkmark \\
& \quad R_g(c_g, 2) \quad \checkmark \\
& \quad \ldots \quad \checkmark \\
& \quad R_g(c_g, N) \quad \checkmark \\
\text{probe} & \quad R_f(c_f, 1, c_f) \quad \checkmark \\
& \quad R_f(c_f, 2, c_f) \quad \checkmark \\
& \quad \ldots \quad \checkmark \\
& \quad R_f(c_f, N, c_f) \quad \checkmark
\end{align*}
\]
Why is relational e-matching faster?

$f(\alpha, g(\alpha))$  

Enum all terms of shape $f(\alpha, g(\beta))$  

Check if $\alpha = \beta$ only before yielding  

$Q(root, \alpha) \leftarrow R_f(root, \alpha, x), R_g(x, \alpha)$  

Build indices on both $\alpha$ and $x$.  

Only enum terms where constraints on both $x$ and $\alpha$ are satisfied.

structural constraints  

equality constraints
Data complexity results (see paper)

**Theorem 9.** Relational e-matching is worst-case optimal; that is, fix a pattern $p$, let $M(p, E)$ be the set of substitutions yielded by e-matching on an e-graph $E$ with $N$ e-nodes, relational e-matching runs in time $O(\max_E(|M(p, E)|))$.

**Theorem 10.** Fix an e-graph $E$ with $N$ e-nodes that compiles to a database $I$, and a fix pattern $p$ that compiles to conjunctive query $Q(X) \leftarrow R_1(X_1), \ldots, R_m(X_m)$. Relational e-matching $p$ on $E$ runs in time $O\left(\sqrt{|Q(I)| \times \prod_i |R_i|}\right) \leq O\left(\sqrt{|Q(I)| \times N^m}\right)$. 
6 orders of magnitude speedup

Relational e-matching: asymptotic speedup

Index building takes time

6 orders of magnitude speedup

Speedup for specific linear patterns ((+ a (* -1 b))

Similar performance on linear patterns.
New Capabilities: *Multi-patterns*

\[
x = \text{matmul}(a, b),
\]

\[
y = \text{matmul}(a, c)
\]

\[
x = \text{split1}(\text{matmul}(a, \text{concat}(b, c))),
\]

\[
y = \text{split2}(\text{matmul}(a, \text{concat}(b, c)))
\]
New Capabilities: *Multi-patterns*

\[ x = \text{matmul}(a, b), \]
\[ y = \text{matmul}(a, c) \]

\[ x = \text{split1(matmul(a,concat(b, c)))}, \]
\[ y = \text{split2(matmul(a,concat(b, c)))} \]

search for two patterns _anywhere_ in the e-graph
New Capabilities: *Multi-patterns*

\[ x = \text{matmul}(a, b), \]
\[ y = \text{matmul}(a, c) \]

\[ x = \text{split1}(\text{matmul}(a, \text{concat}(b, c))), \]
\[ y = \text{split2}(\text{matmul}(a, \text{concat}(b, c))) \]

- search for two patterns *anywhere* in the e-graph
- perform two merges, each on a separate e-class!
egg EqSat Toolkit [POPL 2021, Distinguished Paper]

✓ Deferred invariant maintenance & batching
✓ Relational e-matching [POPL 2022]
❏ E-class analyses
❏ Rewrite rule synthesis with Ruler [OOPSLA 2021, Distinguished Paper]

Applications
- 3D CAD in Szalinski, FP Accuracy in Herbie, Lib Learning in Babble, …
- EVM simplify @ Certora, wasm JIT @ Fastly, datapath optimize @ Intel, …
Syntactic rewriting is not enough…

● How many rules do we need for constant folding?
  ○ 2 + 2 → 4, 3 + 4 → 6, 4 + 6 → 10, … a lot!

● What about satisfying guards for conditional rules?
  ○ x / x → 1 only ok if x <> 0

● In general, many optimizations depend on analyses!
  ○ nullability, tensor shape, intervals, free variables, …
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!
E-class analyses

- One 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)
E-class analysis invariant

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

- for each e-class
- fixed point
- Analysis data is LUB (lattice properties)
Program analysis modulo equivalence

- Tightest summary over all equivalent represented terms!

To demonstrate an advantage of this approach, consider the following example, for $x \in [0, 1], y \in [1, 2]$, where the following concrete-equivalences are discovered via rewriting:

$$1 - \frac{2y}{x + y} \quad \in \left[-3, \frac{1}{3}\right] \quad (5)$$

$$\approx \frac{x - y}{x + y} \quad \in [-2, 0] \quad (6)$$

$$\approx \frac{2x}{x + y} - 1 \quad \in [-1, 1]. \quad (7)$$

The interval associated with the e-class containing these three expressions is $[-3, \frac{1}{3}] \cap [-2, 0] \cap [-1, 1] = [-1, 0]$. We

Sam Coward et al. (2022)
Program analysis modulo equivalence

- Tightest summary over all equivalent represented terms!

- Virtuous cycle: facts enable rewrites, rewrites improve facts!

\[
\begin{align*}
\text{(a) Initial e-graph: } & \frac{y}{1+y} \\
\text{(b) Applying } & \frac{y}{1+y} \rightarrow \frac{1}{1+y}
\end{align*}
\]

\[
\begin{align*}
\text{containing these three expressions is } & [-3, \frac{1}{3}] \cap [-2, 0] \cap [-1, 1] = [-1, 0].
\end{align*}
\]

Sam Coward et al. (2022)
egg EqSat Toolkit [POPL 2021, Distinguished Paper]

✓ Deferred invariant maintenance & batching
✓ Relational e-matching [POPL 2022]
✓ E-class analyses
❑ Rewrite rule synthesis with Ruler [OOPSLA 2021, Distinguished Paper]
❑ Applications
  ❑ 3D CAD in Szalinski, FP Accuracy in Herbie, Lib Learning in Babble, …
  ❑ EVM simplify @ Certora, wasm JIT @ Fastly, datapath optimize @ Intel, …
Chicken & Egg
EqSat and egg can only be as good as user’s rules…
EqSat and egg can only be as good as user’s rules...

Where do rules come from?

- Typically hand written by experts
- Time consuming, often takes years
- Too few / too many / unsound rules
A 3-step approach for inferring rewrite rules

A 3-step approach for inferring rewrite rules

Enumerate terms from a grammar

\[ a, b, 0, +, \ldots \]

A 3-step approach for inferring rewrite rules

Enumerate terms from a grammar

Find candidates: interpret over concrete inputs

“Fingerprints”

A 3-step approach for inferring rewrite rules

Enumerate terms from a grammar

Find candidates: interpret over concrete inputs

(x + y) ⇄ (y + x)

A 3-step approach for inferring rewrite rules

Enumerate terms from a grammar

Find candidates: interpret over concrete inputs

"Fingerprints"

\[ a, b, 0, +, \ldots \]

\[ + + + + \ldots \]

\[ + + + + + \ldots \]

\[ + + + + + \ldots \]

\[ + + + + + \ldots \]

\[ a \quad b \quad 0 \]

\[ (x + 0) \leftrightarrow x \]
A 3-step approach for inferring rewrite rules

Enumerate terms from a grammar:

\[ a, b, 0, +, \ldots \]

Find candidates: interpret over concrete inputs:

“Fingerprints”

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

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

A 3-step approach for inferring rewrite rules

1. Enumerate terms from a grammar
   - a, b, 0, +, ...

2. Find candidates: interpret over concrete inputs
   - "Fingerprints"

3. Filter candidates to get final ruleset
   - Remove redundant rules
     - x + O ↔ O + x
     - y + O ↔ O + y
     - x + y ↔ y + x
A 3-step approach for inferring rewrite rules

1. Enumerate terms from a grammar
   - Exponentially many terms!

2. Find candidates: interpret over concrete inputs
   - Too many candidates, some potentially unsound!

3. Filter candidates to get final ruleset
   - Hard to find a small, useful ruleset

A 3-step approach for inferring rewrite rules

Inferring **Small, Useful** Rule sets **Faster**
using Equality Saturation!

Equality Saturation for not just applying rewrites, but also *inferring* them!
**Ruler**

**Grammar**
- \( e ::= x, 0, e + e, e * e, \ldots \)  

**Interpreter**
- `match e {  
  | const => const  
  | var (v) => lookup (v)  
  | e1 + e2 => eval(e1) + eval(e2)  
  | e1 * e2 => eval(e1) * eval(e2)  
  | \ldots  
}  

**Validator**
- SMT / model check / fuzz

**Term Enumeration**
- Modulo Equivalence

**Candidate Rule Generation**

**Rule Selection**

**Rewrites**
- \( x + 0 = x \)  
- \( x * 1 = x \)  
- \( x - 0 = x \)  
- \( x / 1 = x \)  
- \( x + y = y + x \)  
- \( x * (y + z) = (x * y) + z \)  
- \( x * (y * z) = (x * y) * z \)  

**Enumeration**

**Candidate Generation**

**Rule Selection**
Ruler

Grammar

\[ e ::= x, 0, e + e, e \times e, \ldots \]

Interpreter

\[
\text{match } e \{ \\
\quad \text{const } \Rightarrow \text{const} \\
\quad \text{var } (v) \Rightarrow \text{lookup } (v) \\
\quad e_1 + e_2 \Rightarrow \text{eval } (e_1) + \text{eval } (e_2) \\
\quad e_1 \times e_2 \Rightarrow \text{eval } (e_1) \times \text{eval } (e_2) \\
\quad \ldots \\
\}
\]

Validator

SMT / model check / fuzz

Enumeration

Candidate Generation

Rule Selection

Rewrites

\[
\begin{align*}
    x + 0 &= x \\
    x \times 1 &= x \\
    x - 0 &= x \\
    x / 1 &= x \\
    x + y &= y + x \\
    x + (y + z) &= (x + y) + z \\
    x \times (y \times z) &= (x \times y) \times z
\end{align*}
\]
Enumeration modulo equality saturation

Exponentially many terms!
Exponentially many terms!

Enumerate over an E-graph

a, b, 0, +, ...

E-classes

Exponentially many terms!

Enumerate over an E-graph
Enumeration modulo equality saturation

Exponentially many terms!

Enumerate over an E-graph

E-classes

Apply current ruleset

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

\[(x + x) + (y + x)\]
Enumeration modulo equality saturation

Exponentially many terms!

Enumerate over an E-graph

Apply current ruleset

Merge equivalent terms

(a + b) + 0 = (b + a) + 0

E-classes

(a + b) + 0 = (b + a) + 0
Exponentially many terms!

Enumerate over an E-graph

Apply current ruleset

Merge equivalent terms

Enumeration modulo equality saturation

Shrinks the term space by applying rewrites as they are learned!
Ruler

Grammar

\[ e ::= x, 0, e + e, e * e, \ldots \]

Interpreter

\[
\text{match } e \{
\text{const } \Rightarrow \text{const}
\text{var } (v) \Rightarrow \text{lookup } (v)
\text{e1 + e2 } \Rightarrow \text{eval } (e1) + \text{eval } (e2)
\text{e1 * e2 } \Rightarrow \text{eval } (e1) * \text{eval } (e2)
\ldots
\}
\]

Validator

SMT / model check / fuzz

Enumeration

Candidate Generation

Rule Selection

Rewrites

\[
\begin{align*}
x + 0 &= x \\
x + 1 &= x \\
x - 0 &= x \\
x / 1 &= x \\
x + y &= y + x \\
x + (y + z) &= (x + y) + z \\
x * (y * z) &= (x * y) * z
\end{align*}
\]
Candidate generation by characteristic vector matching

Seed initial E-classes with concrete values (cvecs) from the domain

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>-2</th>
<th>7</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>-2</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>-7</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Candidate generation by characteristic vector matching

Seed initial E-classes with concrete values (cvecs) from the domain

Compute the cvecs for newly enumerated E-classes

Seed initial E-classes with concrete values (cvecs) from the domain
Candidate generation by characteristic vector matching

Seed initial E-classes with concrete values (cvecs) from the domain

Compute the cvecs for newly enumerated E-classes

(x + y)  ↔  (y + x)
Candidate generation by characteristic vector matching

Seed initial E-classes with concrete values (cvecs) from the domain

Compute the cvecs for newly enumerated E-classes

Candidate generation by characteristic vector matching
Candidate generation by characteristic vector matching

Seed initial E-classes with concrete values (cvecs) from the domain

Compute the cvecs for newly enumerated E-classes

Validate candidates using SMT, fuzzing, model checking
Ruler

Grammar

\[ e ::= x, 0, e + e, e * e, \ldots \]

Interpreter

\[
\text{match } e \{ \\
\quad \text{const } \Rightarrow \text{const} \\
\quad \text{var } (v) \Rightarrow \text{lookup } (v) \\
\quad e1 + e2 \Rightarrow \text{eval } (e1) + \text{eval } (e2) \\
\quad e1 * e2 \Rightarrow \text{eval } (e1) * \text{eval } (e2) \\
\quad \ldots \\
\}
\]

Validator

SMT / model check / fuzz

Term Enumeration Modulo Equivalence

Candidate Rule Generation

Rule Selection

Rewrites

\[
\begin{align*}
  x + 0 &= x \\
  x * 1 &= x \\
  x - 0 &= x \\
  x / 1 &= x \\
  x + y &= y + x \\
  x + (y + z) &= (x + y) + z \\
  x * (y * z) &= (x * y) * z 
\end{align*}
\]

Enumeration

Candidate Generation

Rule Selection
Rule selection with equality saturation

\[ C = \]

\[
\begin{align*}
(x + y) & \iff (y + x) \\
(x + 0) & \iff (0 + x) \\
(y + 0) & \iff (0 + y) \\
(x \cdot y) & \iff (y \cdot x) \\
(x \cdot 1) & \iff (1 \cdot x) \\
(y \cdot 1) & \iff (1 \cdot y)
\end{align*}
\]
Rule selection with equality saturation

C =

\[
\begin{align*}
(x + y) & \iff (y + x) \\
(x \times y) & \iff (y \times x) \\
(x + 0) & \iff (0 + x) \\
(y + 0) & \iff (0 + y) \\
(x \times 1) & \iff (1 \times x) \\
(y \times 1) & \iff (1 \times y)
\end{align*}
\]

Rank sound candidates based on generality and pick top-k (2)
Rule selection with equality saturation

Rank sound candidates based on generality and pick top-k (2)

Instantiate and add to rule E-graph
Rule selection with equality saturation

Rank sound candidates based on generality and pick top-k (2)

Instantiate and add to rule E-graph

\[
\begin{align*}
(x + y) & \iff (y + x) \\
(x \ast y) & \iff (y \ast x) \\
(x + 0) & \iff (0 + x) \\
(y + 0) & \iff (0 + y) \\
(x \ast 1) & \iff (1 \ast x) \\
(y \ast 1) & \iff (1 \ast y)
\end{align*}
\]
Rule selection with equality saturation

Instantiate and add to rule E-graph

Run equality saturation
Rule selection with equality saturation

All four rules are redundant and therefore discarded!

Instantiate and add to rule E-graph

Run equality saturation
Rule selection with equality saturation

Continue processing until candidate set is empty or has only unsound ones left!

All four rules are redundant and therefore discarded!

Instantiate and add to rule E-graph

Run equality saturation
Rule selection with equality saturation

Larger top-k makes Ruler faster
Smaller top-k gives smaller rulesets
See paper for detailed comparison!

\[
\begin{align*}
(x + O) &\iff (O + x) \\
(y + O) &\iff (O + y) \\
(x \ast l) &\iff (l \ast x) \\
(y \ast l) &\iff (l \ast y)
\end{align*}
\]
Rule selection with equality saturation

Larger top-k makes Ruler faster

Smaller top-k gives smaller rulesets

See paper for detailed comparison!

---

Shrinks the candidate space by applying rewrites as they are learned!
Ruler

Grammar

```
e ::= x, 0, e + e, e * e, ...
```

Interpreter

```
match e {
    const => const
    var (v) => lookup (v)
    e1 + e2 => eval (e1) + eval(e2)
    e1 * e2 => eval (e1) * eval(e2)
} ...
```

Validator

SMT / model check / fuzz

Term Enumeration Modulo Equivalence

Candidate Rule Generation

Rule Selection

Rewrites

```
x + 0 = x
x * 1 = x
x - 0 = x
x / 1 = x
x + y = y + x
x + (y + z) = (x + y) + z
x * (y * z) = (x * y) * z
```
Equality saturation “soundiness”

Equality Saturation *amplifies* unsoundness!
Equality saturation “soundiness”

Equality Saturation *amplifies* unsoundness!
Equality saturation “soundiness”

Equality Saturation *amplifies* unsoundness!

current ruleset

\[(y \ast 0) \leftrightarrow 0\]
\[(y \ast 0) \leftrightarrow 1\]
Equality saturation "soundiness"

Equality Saturation *amplifies* unsoundness!

- Run equality saturation on term E-graph
Equality saturation “soundiness”

Equality Saturation *amplifies* unsoundness!

Run equality saturation on current ruleset term E-graph
Equality saturation “soundiness”

Equality Saturation amplifies unsoundness!

Run equality saturation on term E-graph

Unsound merge, 0 != 1
Implementation

https://github.com/uwplse/ruler

Implemented in Rust

Uses egg for equality saturation
Evaluation

Ruler vs Other tools (CVC4)
How do the rulesets compare?
## Comparison with CVC4

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Domain</th>
<th># Conn</th>
<th>Time (s)</th>
<th># Rules</th>
<th>Drv</th>
<th>Time (s)</th>
<th># Rules</th>
<th>Drv</th>
<th>Time</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>bool</td>
<td>2</td>
<td>0.01</td>
<td>20</td>
<td>1</td>
<td>0.13</td>
<td>53</td>
<td>1</td>
<td>0.06</td>
<td>0.38</td>
</tr>
<tr>
<td></td>
<td>bool</td>
<td>3</td>
<td>0.06</td>
<td>28</td>
<td>1</td>
<td>0.82</td>
<td>293</td>
<td>1</td>
<td>0.07</td>
<td>0.10</td>
</tr>
<tr>
<td></td>
<td>bv4</td>
<td>2</td>
<td>0.14</td>
<td>49</td>
<td>1</td>
<td>4.47</td>
<td>135</td>
<td>0.98</td>
<td>0.03</td>
<td>0.36</td>
</tr>
<tr>
<td></td>
<td>bv4</td>
<td>3</td>
<td>4.30</td>
<td>272</td>
<td>1</td>
<td>372.26</td>
<td>1978</td>
<td>1</td>
<td>0.01</td>
<td>0.14</td>
</tr>
<tr>
<td></td>
<td>bv32</td>
<td>2</td>
<td>13.00</td>
<td>46</td>
<td>0.97</td>
<td>18.53</td>
<td>126</td>
<td>0.93</td>
<td>0.70</td>
<td>0.37</td>
</tr>
<tr>
<td></td>
<td>bv32</td>
<td>3</td>
<td>630.09</td>
<td>188</td>
<td>0.98</td>
<td>1199.53</td>
<td>1782</td>
<td>0.91</td>
<td>0.53</td>
<td>0.11</td>
</tr>
</tbody>
</table>
## Comparison with CVC4

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Domain</th>
<th># Conn</th>
<th>Ruler</th>
<th>CVC4</th>
<th>Ruler / CVC4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Time (s)</td>
<td># Rules</td>
<td>Drv</td>
</tr>
<tr>
<td>bool</td>
<td>2</td>
<td></td>
<td>0.01</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>bool</td>
<td>3</td>
<td></td>
<td>0.06</td>
<td>28</td>
<td>1</td>
</tr>
<tr>
<td>bv4</td>
<td>2</td>
<td></td>
<td>0.14</td>
<td>49</td>
<td>1</td>
</tr>
<tr>
<td>bv4</td>
<td>3</td>
<td></td>
<td>4.30</td>
<td>272</td>
<td>1</td>
</tr>
<tr>
<td>bv32</td>
<td>2</td>
<td></td>
<td>13.00</td>
<td>46</td>
<td>0.97</td>
</tr>
<tr>
<td>bv32</td>
<td>3</td>
<td></td>
<td>630.09</td>
<td>188</td>
<td>0.98</td>
</tr>
</tbody>
</table>
## Comparison with CVC4

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Domain</th>
<th># Conn</th>
<th>Time (s)</th>
<th># Rules</th>
<th>Drv</th>
<th>Time (s)</th>
<th># Rules</th>
<th>Drv</th>
<th>Time</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>bool</td>
<td>2</td>
<td>0.01</td>
<td>20</td>
<td>1</td>
<td>0.13</td>
<td>53</td>
<td>1</td>
<td>0.06</td>
<td>0.38</td>
</tr>
<tr>
<td></td>
<td>bool</td>
<td>3</td>
<td>0.06</td>
<td>28</td>
<td>1</td>
<td>0.82</td>
<td>293</td>
<td>1</td>
<td>0.07</td>
<td>0.10</td>
</tr>
<tr>
<td></td>
<td>bv4</td>
<td>2</td>
<td>0.14</td>
<td>49</td>
<td>1</td>
<td>4.47</td>
<td>135</td>
<td>0.98</td>
<td>0.03</td>
<td>0.36</td>
</tr>
<tr>
<td></td>
<td>bv4</td>
<td>3</td>
<td>4.30</td>
<td>272</td>
<td>1</td>
<td>372.26</td>
<td>1978</td>
<td>1</td>
<td>0.01</td>
<td>0.14</td>
</tr>
<tr>
<td></td>
<td>bv32</td>
<td>2</td>
<td>13.00</td>
<td>46</td>
<td>0.97</td>
<td>18.53</td>
<td>126</td>
<td>0.93</td>
<td>0.70</td>
<td>0.37</td>
</tr>
<tr>
<td></td>
<td>bv32</td>
<td>3</td>
<td>630.09</td>
<td>188</td>
<td>0.98</td>
<td>1199.53</td>
<td>1782</td>
<td>0.91</td>
<td>0.53</td>
<td>0.11</td>
</tr>
</tbody>
</table>
## Comparison with CVC4

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Ruler</th>
<th>CVC4</th>
<th>Ruler / CVC4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (s)</td>
<td># Rules</td>
<td>Drv</td>
</tr>
<tr>
<td>bool</td>
<td>0.01</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>bool</td>
<td>0.06</td>
<td>28</td>
<td>1</td>
</tr>
<tr>
<td>bv4</td>
<td>0.14</td>
<td>49</td>
<td>1</td>
</tr>
<tr>
<td>bv4</td>
<td>4.30</td>
<td>272</td>
<td>1</td>
</tr>
<tr>
<td>bv32</td>
<td>13.00</td>
<td>46</td>
<td>0.97</td>
</tr>
<tr>
<td>bv32</td>
<td>630.09</td>
<td>188</td>
<td>0.98</td>
</tr>
</tbody>
</table>
## Comparison with CVC4

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Ruler</th>
<th>CVC4</th>
<th>Ruler / CVC4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (s)</td>
<td># Rules</td>
<td>Drv</td>
</tr>
<tr>
<td>Domain</td>
<td># Conn</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bool</td>
<td>2</td>
<td>0.01</td>
<td>20</td>
</tr>
<tr>
<td>bool</td>
<td>3</td>
<td>0.06</td>
<td>28</td>
</tr>
<tr>
<td>bv4</td>
<td>2</td>
<td>0.14</td>
<td>49</td>
</tr>
<tr>
<td>bv4</td>
<td>3</td>
<td>4.30</td>
<td>272</td>
</tr>
<tr>
<td>bv32</td>
<td>2</td>
<td>13.00</td>
<td>46</td>
</tr>
<tr>
<td>bv32</td>
<td>3</td>
<td>630.09</td>
<td>188</td>
</tr>
</tbody>
</table>

Fraction of the 1782 rules from CVC4 that the 188 rules from Ruler can derive via equality saturation
Comparison with CVC4

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Ruler</th>
<th>CVC4</th>
<th>Ruler / CVC4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (s)</td>
<td># Rules</td>
<td>Drv</td>
</tr>
<tr>
<td>bool</td>
<td>0.01</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>bool</td>
<td>0.06</td>
<td>28</td>
<td>1</td>
</tr>
<tr>
<td>bv4</td>
<td>0.14</td>
<td>49</td>
<td>1</td>
</tr>
<tr>
<td>bv4</td>
<td>4.30</td>
<td>272</td>
<td>1</td>
</tr>
<tr>
<td>bv32</td>
<td>13.00</td>
<td>46</td>
<td>0.97</td>
</tr>
<tr>
<td>bv32</td>
<td>630.09</td>
<td>188</td>
<td>0.98</td>
</tr>
</tbody>
</table>

Ruler infers a smaller, useful ruleset faster
Evaluation

Ruler vs Other tools (CVC4)
How do the rule sets compare?

Ruler vs Humans (Herbie)
Can Ruler compete with experts?
Comparison with human-written rules

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

Herbie detects inaccurate expressions and finds more accurate replacements. The red expression is inaccurate when \( x > 1 \); Herbie's replacement, in blue, is accurate for all \( x \).
Comparison with human-written rules

52 rational rules, designed by the developers over 6 years

55 / 155 benchmarks are purely over rational arithmetic
Comparison with human-written rules

52 rational rules, designed by the developers over 6 years

55 / 155 benchmarks are purely over rational arithmetic

Herbie can generate more-complex expressions that aren't more precise #261
Comparison with human-written rules

52 rational rules, designed by the developers over 6 years

55 / 155 benchmarks are purely over rational arithmetic

Herbie can generate more-complex expressions that aren't more precise

x * y ≟ | x | * | y |

x * x ≟ x * x

Discovered by Ruler, resolved the GitHub issue!
End-to-end: rational Herbie

None:    Remove all rules
Herbie:  Herbie without any changes
Ruler:   Herbie with Ruler’s rules
Both:    Herbie with both original and Ruler’s rules
Rational Herbie: comparing accuracy

None: Remove all rules
Herbie: Herbie without any changes
Ruler: Herbie with Ruler’s rules
Both: Herbie with both original and Ruler’s rules

Ruler’s rules are at least as good as the original Herbie rules
Rational Herbie: comparing AST size

None: Remove all rules
Herbie: Herbie without any changes
Ruler: Herbie with Ruler’s rules
Both: Herbie with both original and Ruler’s rules

Ruler’s rules are at least as good as the original Herbie rules
Rational Herbie: comparing AST size

None: Remove all rules
Herbie: Herbie without any changes
Ruler: Herbie with Ruler’s rules
Both: Herbie with both original and Ruler’s rules

Ruler’s rules are at least as good as the original Herbie rules

See paper for more results!
Rewrite Rule Inference Using Equality Saturation

Equality Saturation improves all three steps!

Ruler: https://github.com/uwplse/ruler
egg EqSat Toolkit [POPL 2021, Distinguished Paper]

✓ Deferred invariant maintenance & batching
✓ Relational e-matching [POPL 2022]
✓ E-class analyses
✓ Rewrite rule synthesis with Ruler [OOPSLA 2021, Distinguished Paper]

Applications

- 3D CAD in Szalinski, FP Accuracy in Herbie, Lib Learning in Babble, …
- EVM simplify @ Certora, wasm JIT @ Fastly, datapath optimize @ Intel, …
Manufacturing is compilation!
Manufacturing is compilation!
Design is programming!
Design is programming!

- CAD
- CSG
- STL
- G-code

portable, BUT difficult to edit
Design is programming!

portable, BUT difficult to edit

decompile?
Szalinski

~1600 LOC, Mesh

~50 LOC, CSG
Szalinski

~1600 LOC, Mesh

~ 50 LOC, CSG
Szalinski [PLDI 2020]
• thousands of models decompiled w/ egg, all < 1 second
Library learning with Babble [POPL 2023]
Library learning with Babble [POPL 2023]
Short Proofs for TV + debugging [FMCAD 2022]

Intel Case Study

Multi-operation circuit optimization and translation validation with egg
4.7 hours -> 2.3 hours

2 inputs

5 outputs

\[ a \ll 1 + \ll 1 \rightarrow 5a + b \]
\[ a \ll 1 \rightarrow 4a + 2b \]
\[ b \ll 1 \rightarrow 3a + 3b \]
\[ b \ll 1 \rightarrow 2a + 4b \]
\[ b \ll 1 \rightarrow a + 5b \]
Short Proofs for TV + debugging [FMCAD 2022]

Intel Case Study

Multi-operation circuit optimization and translation validation with egg

4.7 hours -> 2.3 hours

2 inputs
egg case studies

- Herbie: floating point
  - 3000x faster
  - shape analysis

- SPORES: linear algebra kernels
  - 1.2-5x better
  - generic library

- Tensat: ML compute graphs
  - 23% better, 48x faster
  - dynamic rewrites

- Szalinski: CAD synthesis
  - 12,000 part eval
  - <1s synthesis

- ..., TVM, Java testing, vectorization, hw/sw co-design, educational problems, ...
egg EqSat Toolkit [POPL 2021, Distinguished Paper]

✓ Deferred invariant maintenance & batching

✓ Relational e-matching [POPL 2022]

✓ E-class analyses

✓ Rewrite rule synthesis with Ruler [OOPSLA 2021, Distinguished Paper]

✓ Applications

✓ 3D CAD in Szalinski, FP Accuracy in Herbie, Lib Learning in Babble, …

✓ EVM simplify @ Certora, wasm JIT @ Fastly, datapath optimize @ Intel, …