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

fast

extensible

our big insights

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

take-home message:
go use it!

$$(a * 2) / 2$$

↓

$$a$$

$$(a * 2) / 2$$

$$==$$

$$a$$

$(a * 2) / 2 \Rightarrow a$

rewrite it!

useful

$(x * y) / z = x * (y / z)$

$x / x = 1$

$x * 1 = x$

not so useful

$x * 2 = x \ll 1$

$x * y = y * x$

$x = x * 1$

$(a * 2) / 2 \Rightarrow a * (2 / 2) \Rightarrow a * 1 \Rightarrow a$

happy path

$(x * y) / z = x * (y / z)$

$x / x = 1$

$x * 1 = x$

(a * 2) / 2 → (a << 1) / 2 ✖ wrong turn

loop

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

a → a * 1 → a * 1 * 1 → ... infinite size

pitfalls

but critical for other inputs

$x * 2 = x << 1$

$x * y = y * x$

$x = x * 1$

$(a * 2) / 2 \rightarrow a$

which rewrite? when?

useful	not so useful
$(x * y) / z = x * (y / z)$	$x * 2 = x << 1$
$x / x = 1$	$x * y = y * x$
$x * 1 = x$	$x = x * 1$

x / x = 1

x * 2 = x << 1

x = x * 1

x = 1 * x

which rewrite? when?

<u>all of them! all the time!</u>

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

x * y = y * x

x / x = 1

x * 2 = x << 1

x = x * 1

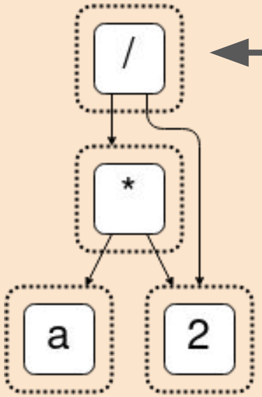
x = 1 * x

equality saturation!

faster!

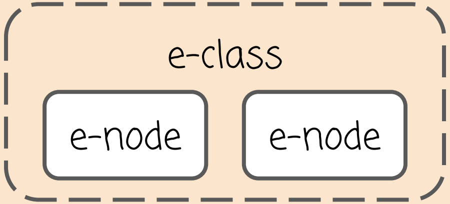
more flexible!

e-graphs?

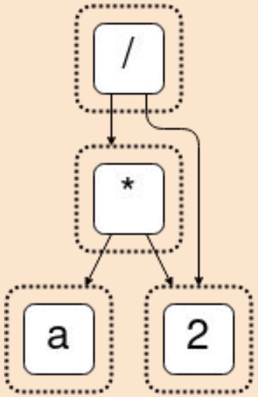


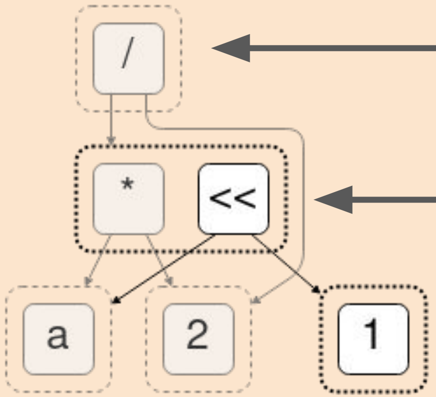
this e-class <u>represents</u>

(a * 2) / 2



growing an e-graph



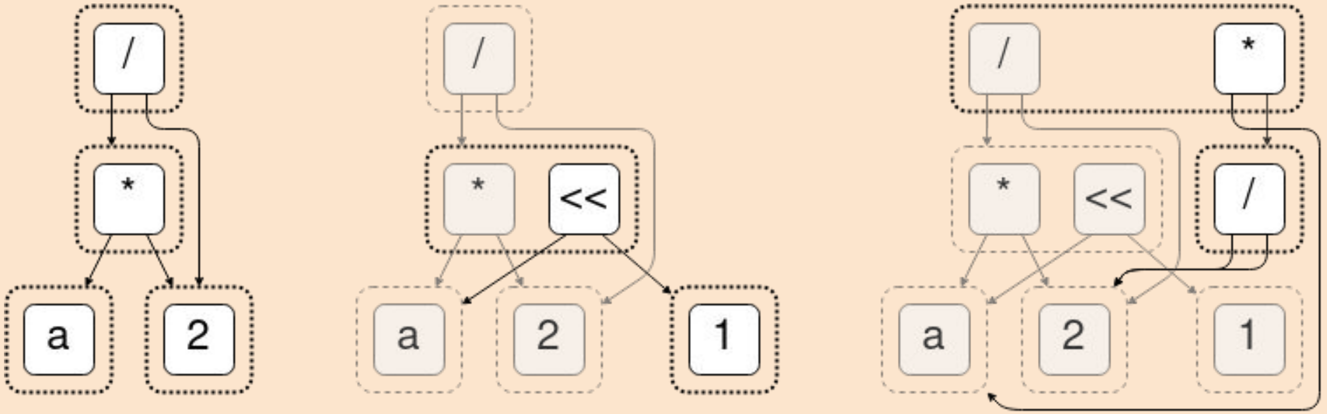


x * 2 → x << 1

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

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

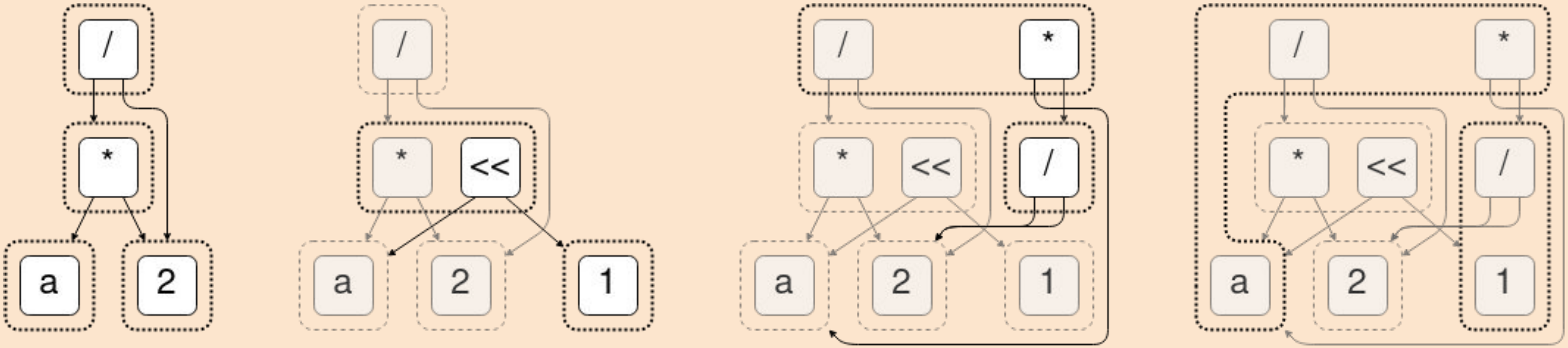
growing an e-graph



x * 2 → x << 1 (x * y) / z → x * (y / z)

e-graphs are compact

a, a * 1,
a * 1 * 1, ...



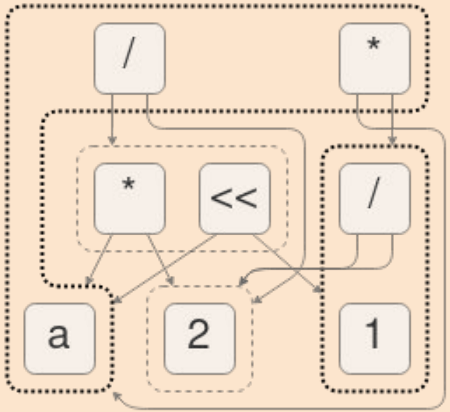
x * 2 → x << 1

(x * y) / z → x * (y / z)

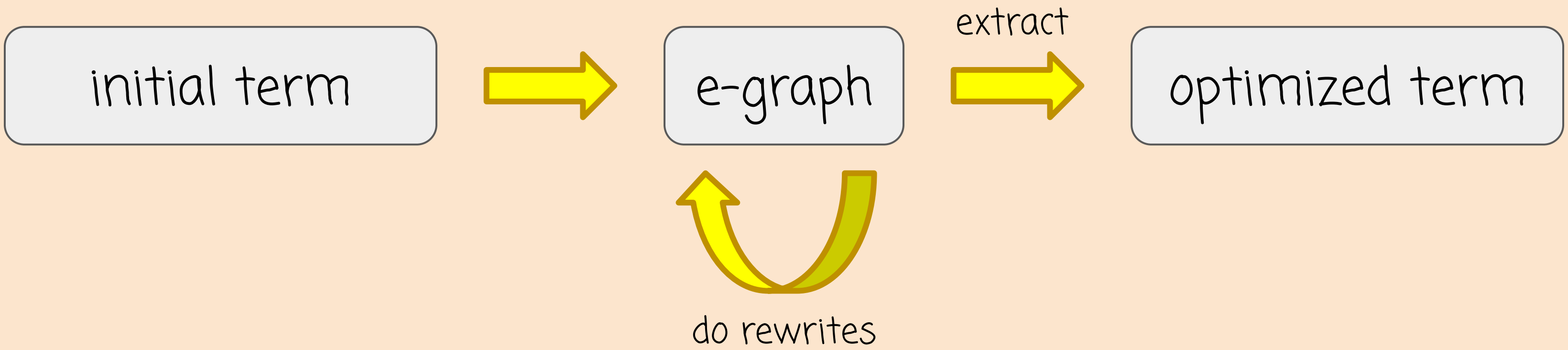
x / x → 1
x * 1 → x

saturation

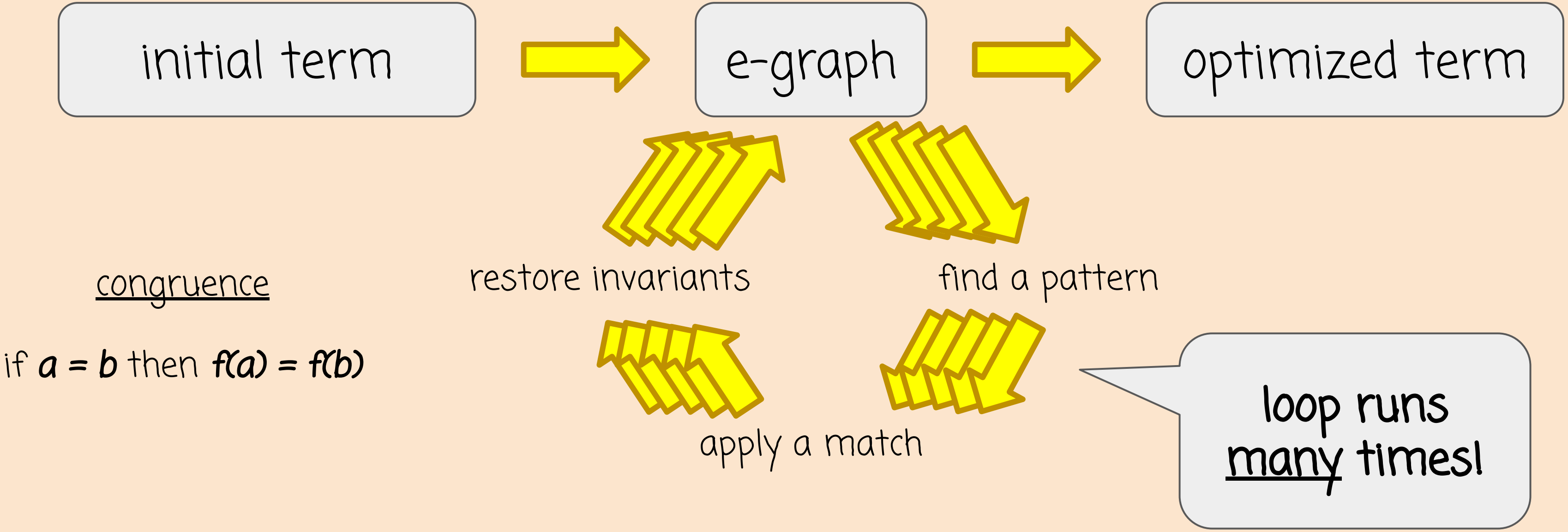
- ✓ $x * 2 \rightarrow x \ll 1$
- ✓ $(x * y) / z \rightarrow x * (y / z)$
- ✓ $x / x \rightarrow 1$
- ✓ $x * 1 \rightarrow x$



equality saturation



equality saturation



talk in a slide

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

our big insights

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

take-home message:
go use it!

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()
```

read

write

restore invariant

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 maintenance
- invariants baked-in

egg's equality saturation

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

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

  return egraph.extract_best()
```

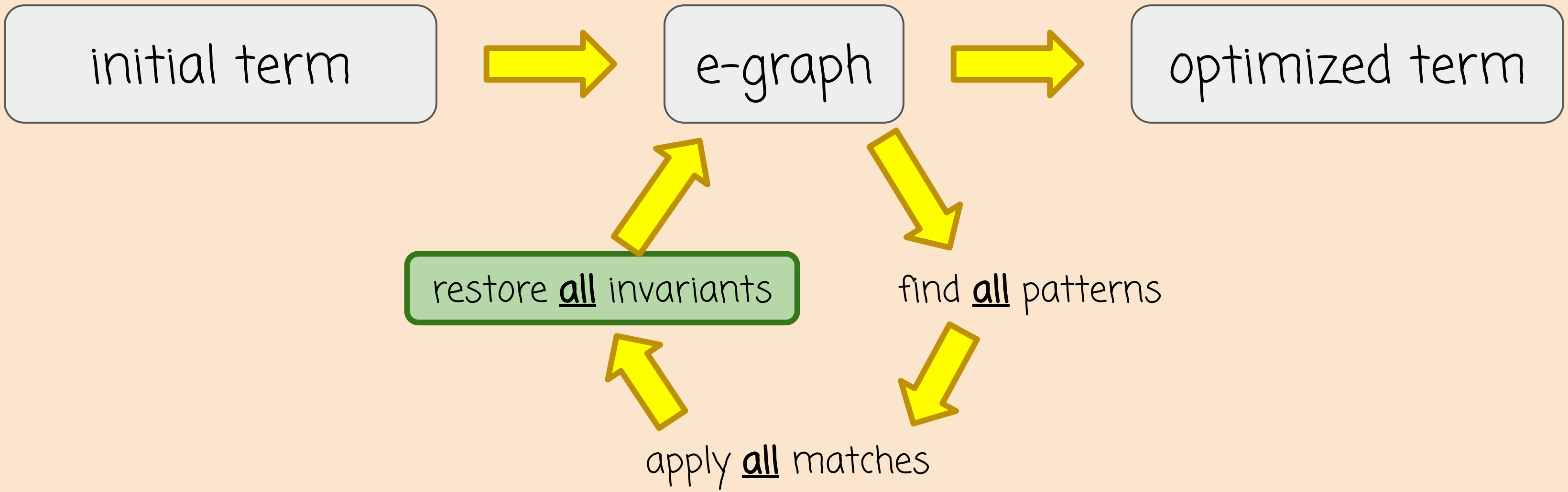
easy to parallelize

invariants broken

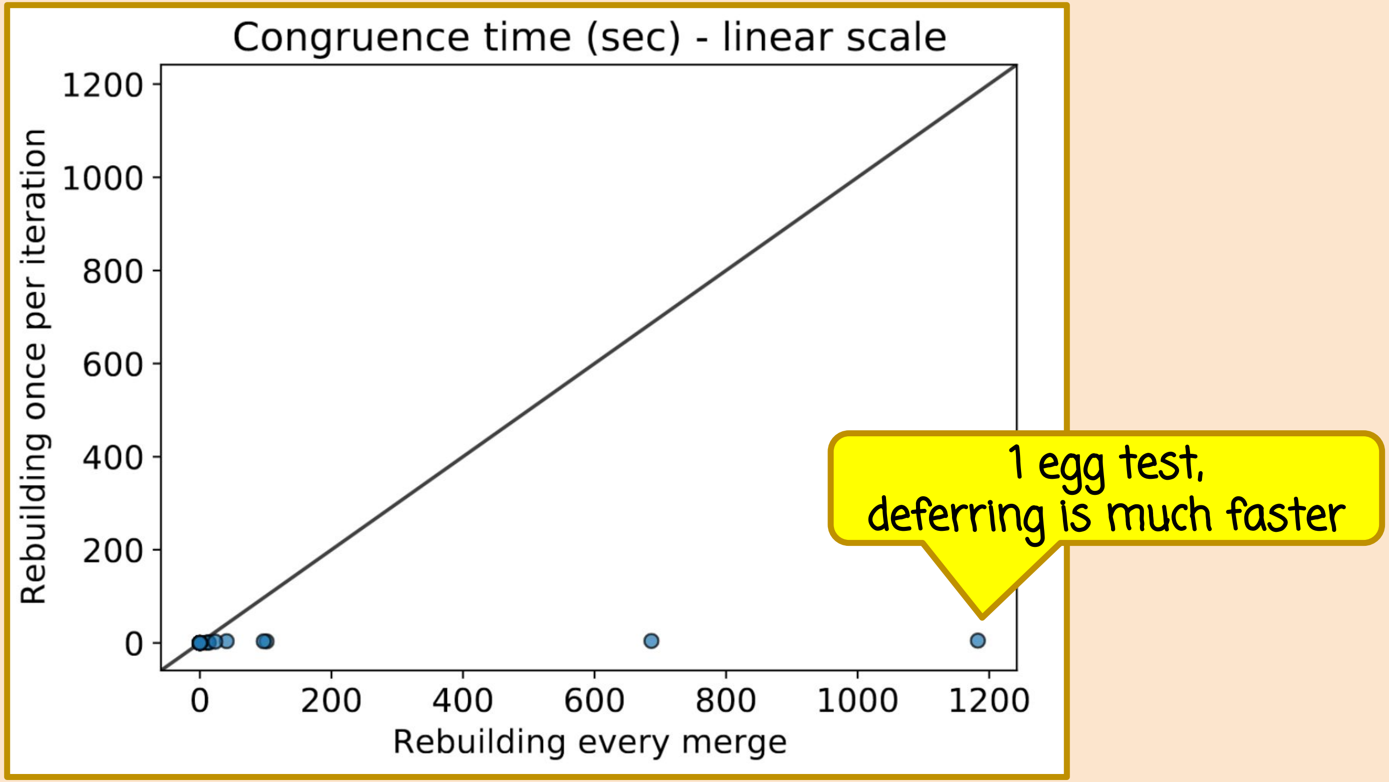
once per iteration

```
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()
```

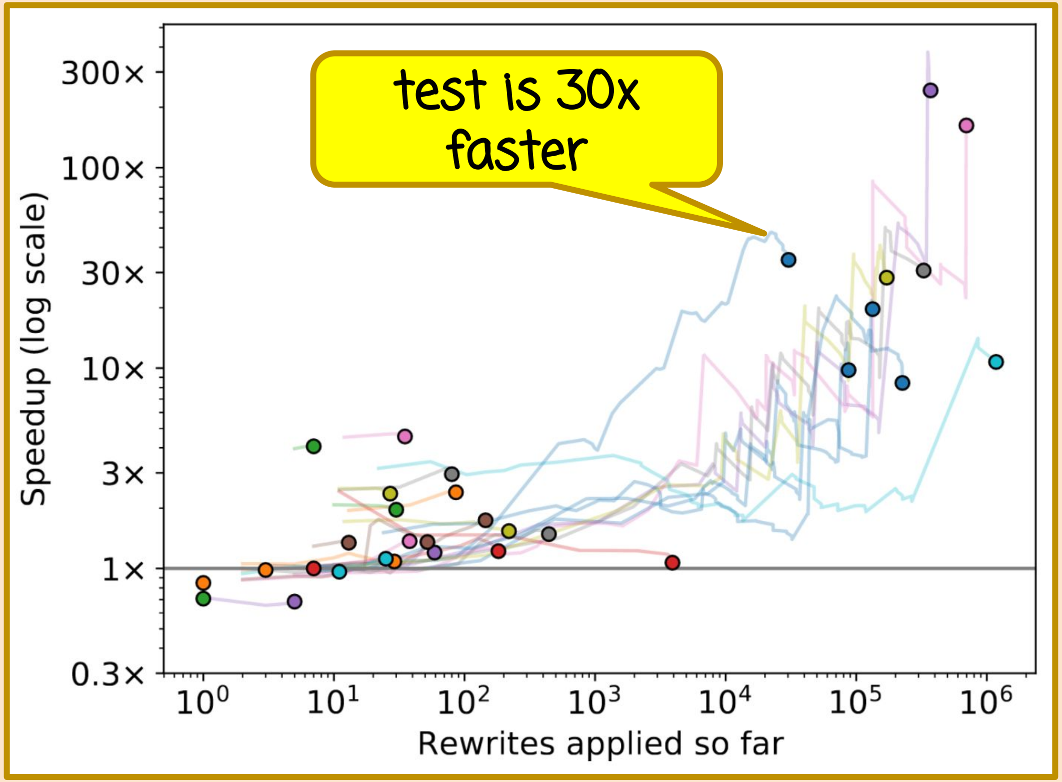
deferred invariant maintenance



rebuilding is faster



rebuilding is faster



why is rebuilding faster?

- consider $f_1(x)$ … $f_n(x)$ and y_1 … y_n
- workload: merge(x, y_1) … 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

our big insights

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

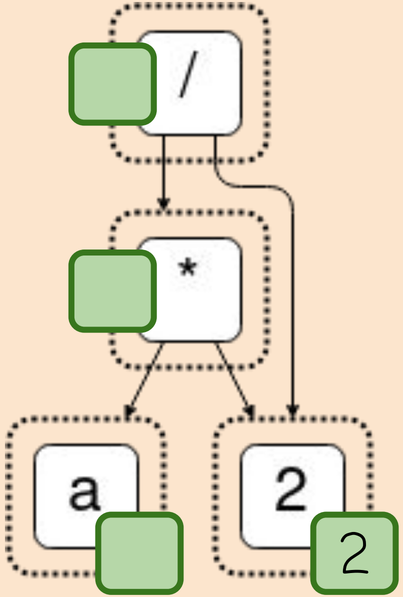
take-home message:
go use it!

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, …

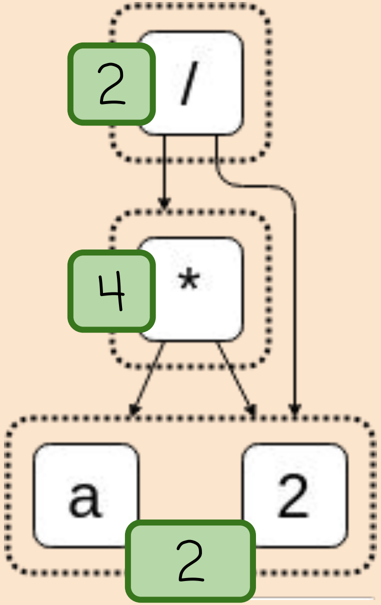
constant folding

- Option<Number> per eclass
- try to eval new e-nodes
- Option "or" on merge



constant folding

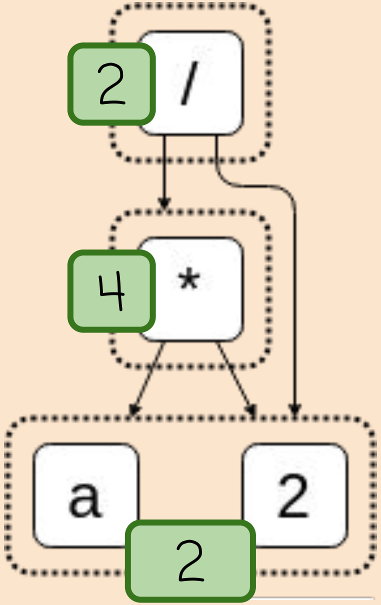
- Option<Number> per eclass
- try to eval new e-nodes
- Option "or" on merge
- it propagates up!



merge(a, 2)

what now?

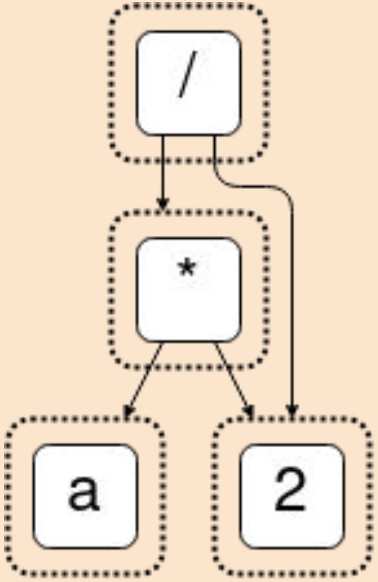
- values are "stuck" in the analysis
- modify hook
- add (or prune) e-nodes



merge(a, 2)

constant folding

- not syntactic
- consider a = 2
- we know a * 2 = 2 * 2 = 4
- facts about children can entail facts about parents

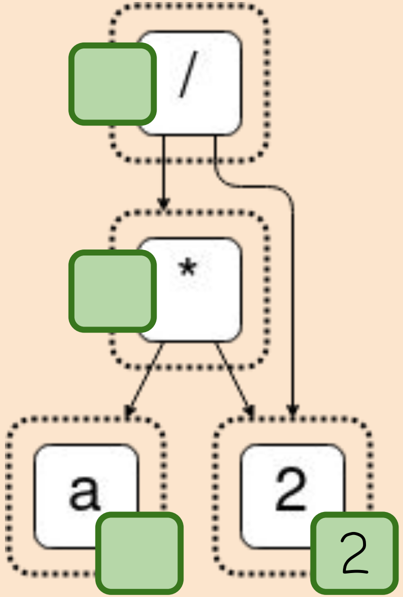


e-class analysis

- 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_{c1}, d_{c2}) → 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



e-class analysis uses

- lift program analyses to e-graphs
- conditional & dynamic rewrites
    - x / x = 1 iff x != 0
- can express other e-graph "hacks"
    - debugging, pruning, on-the-fly extraction

e-class analysis invariant

for each e-class

fixed point

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

Analysis data is LUB (lattice properties)

talk in a slide

- e-graphs and eqsat were always cool

- deferred invariant maintenance
- e-class analyses

our big insights

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

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
- 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 — batching — 3000x faster
- SPORES: linear algebra kernels — shape analysis — 1.2-5x better
- ML compute graphs — generic library — 23% better, 48x faster
- Szalinski: CAD synthesis — dynamic rewrites — 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!