Relational E-Matching

Simpler, Faster, and Optimal

Yihong Zhang, University of Washington
PLDI 2021 Student Research Competition

E-Graphs are everywhere!

Spores [VLDB '20]

Szalinski [PLDI '20]

Herbie [PLDI '15]

TenSat [MLSys '21]

Diospyoros [ASPLOS '21]

Glenside [MAPS '21]

egg [POPL '20]

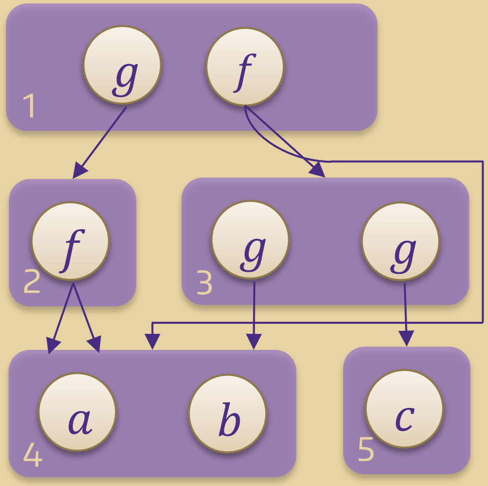
Z3

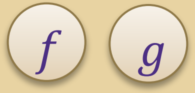
CVC4

Metatheory.jl

...

E-Graphs





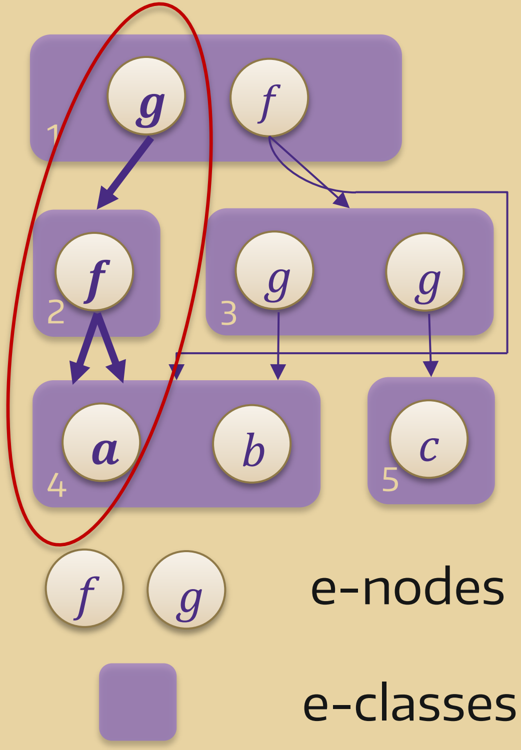
e-nodes

e-classes

e-class 1 represents

$g(f(a,a))$

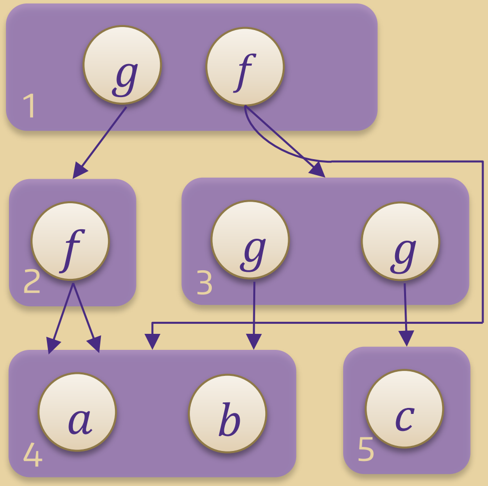
E-Graphs

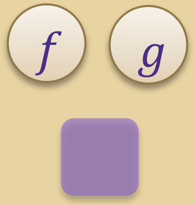


e-class 1 represents

$$g(f(a,a))$$

E-Graphs





e-nodes

e-classes

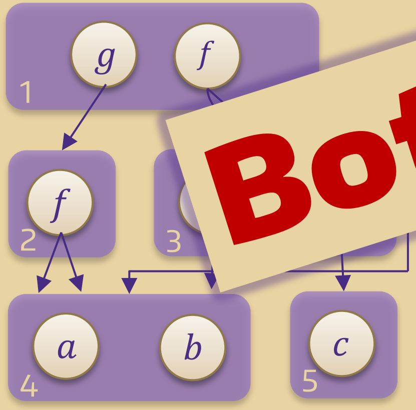
e-class 1 represents

$g(f(a,a))$ $f(a,g(a))$
$g(f(a,b))$ $f(a,g(b))$
$g(f(b,a))$ $f(b,g(a))$
$g(f(b,b))$ $f(b,g(b))$
$f(a,g(c))$
$f(b,g(c))$

exponentially many terms!

E-Matching

- E-matching is the query that finds patterns in an e-graph.



Bottleneck!

responds to

$f(a, g(a))$, $f(a, g(b))$
$f(b, g(a))$, $f(b, g(b))$.

All witnessed by subst. $\{\alpha \longmapsto 4\}$.

- Responsible for 60-90% of the run time.

Existing E-Matching Algorithms

$f(\alpha, g(\alpha))$ ⇒

for all e-class c **in** e-graph E:
👉 **for** all f node n_1 **in** c:
  subst $= \{\alpha \mapsto n_1.child_1, \mathbf{root} \mapsto c\}$
  👉 **for** all g node n_2 **in** $n_1.child_2$:
    if subst$[\alpha] = n_2.child_1$:
      yield subst

Quadratic runtime!

Yet at most $O(N)$ terms.

Relational E-Matching

Relational queries only involving joins e.g.,
$Q(a,c) \leftarrow R(a,b), S(b,c)$

E-Matching

E-Graphs

$\subseteq$

Conjunctive Queries

Relational Databases

Simpler

Comparison to Existing E-Matching

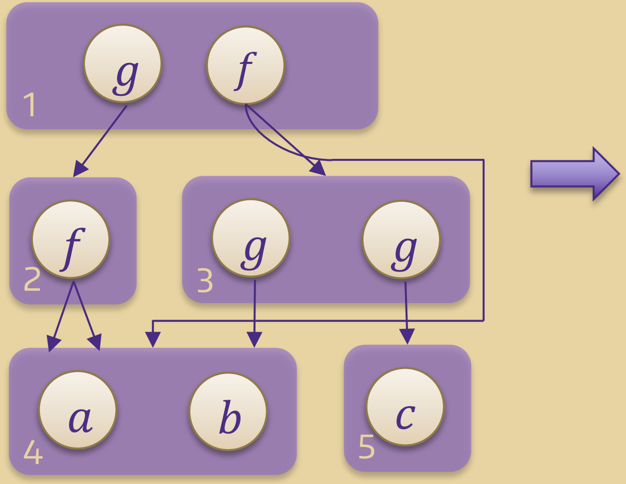
Existing E-Matching

- ✗ Top-down backtracking search only.
- ✗ Exploits structural constraints only.
- ✗ No theoretical guarantee.

Relational E-Matching

- ✓ Top-down, bottom-up, middle-out, etc. depending on the query optimizer.
- ✓ Exploits both structural constraints and equality constraints.
- ✓ Achieves optimality by adapting results from database research.

E-Graphs → Relational Databases



Relations representing function symbols

R_f

eclass-id	$child_1$	$child_2$

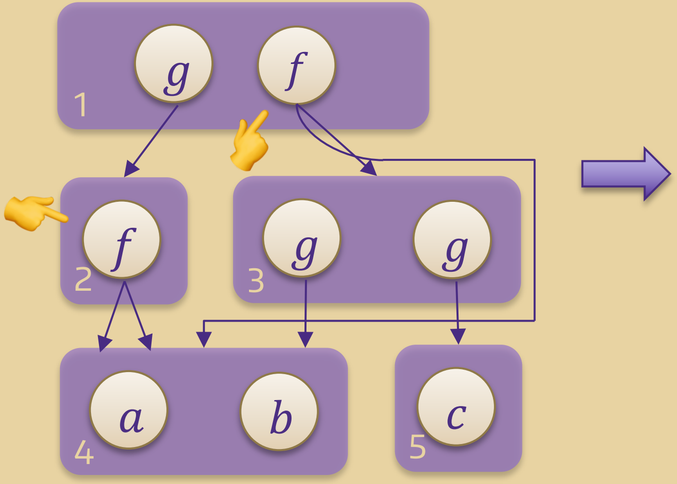
R_g

eclass-id	$child_1$

R_a, R_b, R_c

eclass-id

E-Graphs → Relational Databases



Relations representing function symbols

R_f

eclass-id	$child_1$	$child_2$
1	4	3
2	4	4

Faster E-Matching → Conjunctive Queries

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

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

for all e-class c **in** e-graph E:
  for all f node n_1 **in** c:
    subst $= \{\alpha \longmapsto n_1.child_1, \textbf{root} \longmapsto c\}$
    for all g node n_2 **in** $n_1.child_2$:
      if subst$[\alpha] = n_2.child_1$:
        yield subst

Quadratic

for all r_f **in** R_f:
  if $(r_f.child_2, r_f.child_1)$ **in** R_g:
    yield $\{\textbf{root} \longmapsto R_f.eclass\text{-}id,$
      $\alpha \longmapsto R_f.child_1\}$

Linear

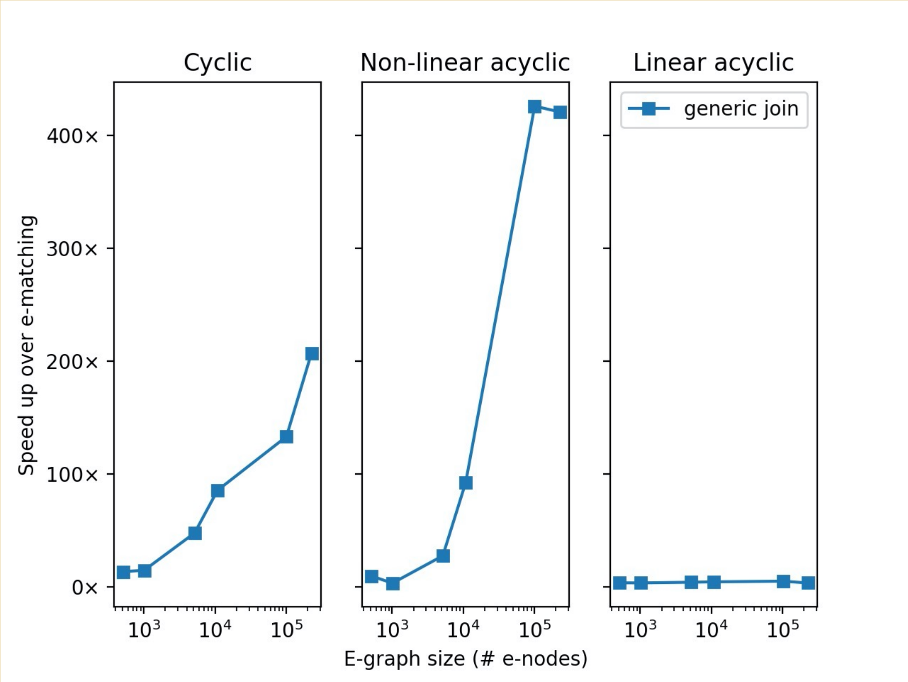
Which CQ Algorithm to Use?

- Traditional two-way join algorithms like hash join and merge-sort join are provably suboptimal.
- We propose to use the generic join algorithm to solve the generated conjunctive query.
- Fix a pattern p, let $M(p, E)$ be the set of substitutions yielded by e-matching on an e-graph E with size n, relational e-matching runs in time

$$\tilde{O}\left(\max_{E}|M(p, E)|\right).$$

Optimal

(Preliminary) Evaluations



Speed-ups over existing e-matching algorithms (de Moura and Bjørner)

- Asymptotic speed-ups on patterns with equality constraints (the cyclic and non-linear acyclic patterns).
- Similar performance on patterns with no equality constraints (the linear pattern).

Thank you!

Why Faster?

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

enumerates all terms of the form $f(\alpha, g(\beta))$, and check if $\boldsymbol{\alpha} = \boldsymbol{\beta}$ only before yielding.

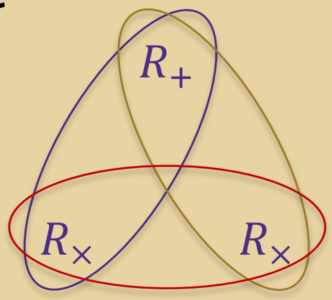
$Q(\text{root}, \alpha) \leftarrow$
$R_f(\text{root}, \alpha, \boldsymbol{x}), R_g(\boldsymbol{x}, \alpha)$

builds indices on both α and $\boldsymbol{x}$; only enumerates terms where constraints on α and $\boldsymbol{x}$ are both satisfied.

Equality constraints are exploited to prune the search space!

Which CQ Algorithm to Use?

- Traditional two-way join algorithms like hash join and merge-sort join are not optimal.
- For example, for conjunctive query whose corresponding hypergraph is cyclic, two-way join algorithms will enumerate asymptotically more atoms than needed.
  - $(\alpha + \beta)\times(\alpha + \gamma)$.



Hypergraph of $(\alpha + \beta)\times(\alpha + \gamma)$

Generic Join Algorithm

> Multi-way join algorithm that avoids enumerating unnecessarily large intermediate relations.

> Worst-case optimal & efficient for both cyclic and acyclic queries.