Relational E-Matching

$$= \subseteq R \bowtie S$$

senior presentation 

Yihong Zhang

ongoing collaboration with  

E-graphs are everywhere!

E-graphs are everywhere!

Spores

Herbie

egg

Glenside

TenSat

Szalinski

Z3

Metatheory.jl

Diospyros

CVC4

E-graphs are everywhere!

Spores

Herbie

egg

Glenside

TenSat

Szalinski

Z3

Metatheory.jl

CVC4

Diospyros

. . .

E-graphs are everywhere!

E-graphs are everywhere!

Equality saturation

- Stores many expressions in a single data structure.
- Uses e-graphs to mitigate the phase ordering problem.

E-graphs are everywhere!

Equality saturation

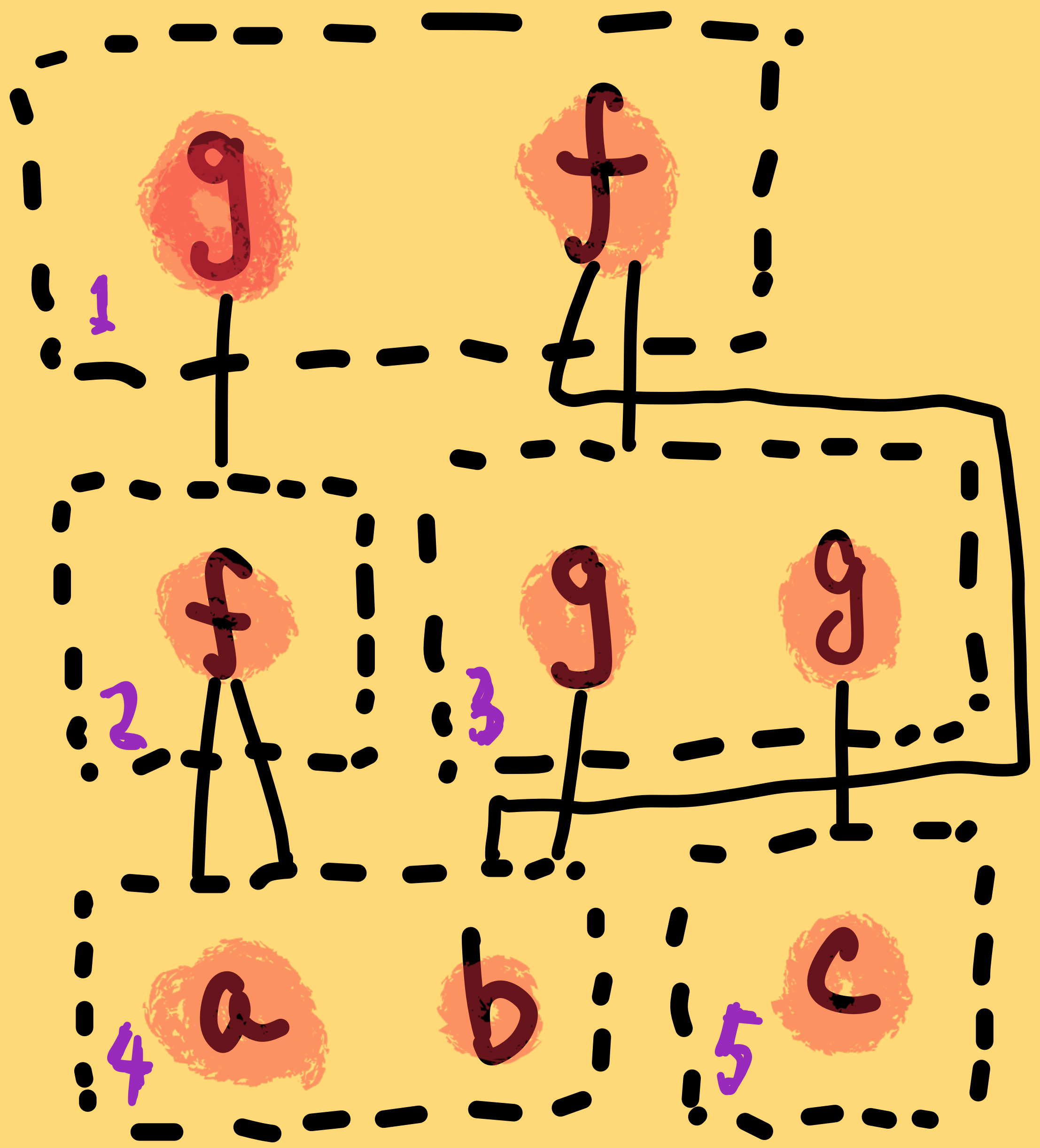
- Stores many expressions in a single data structure.
- Uses e-graphs to mitigate the phase ordering problem.

SMT Solver

- Used in the solver for the theory of equality with uninterpreted functions.
- The "glue solver"

E-graphs

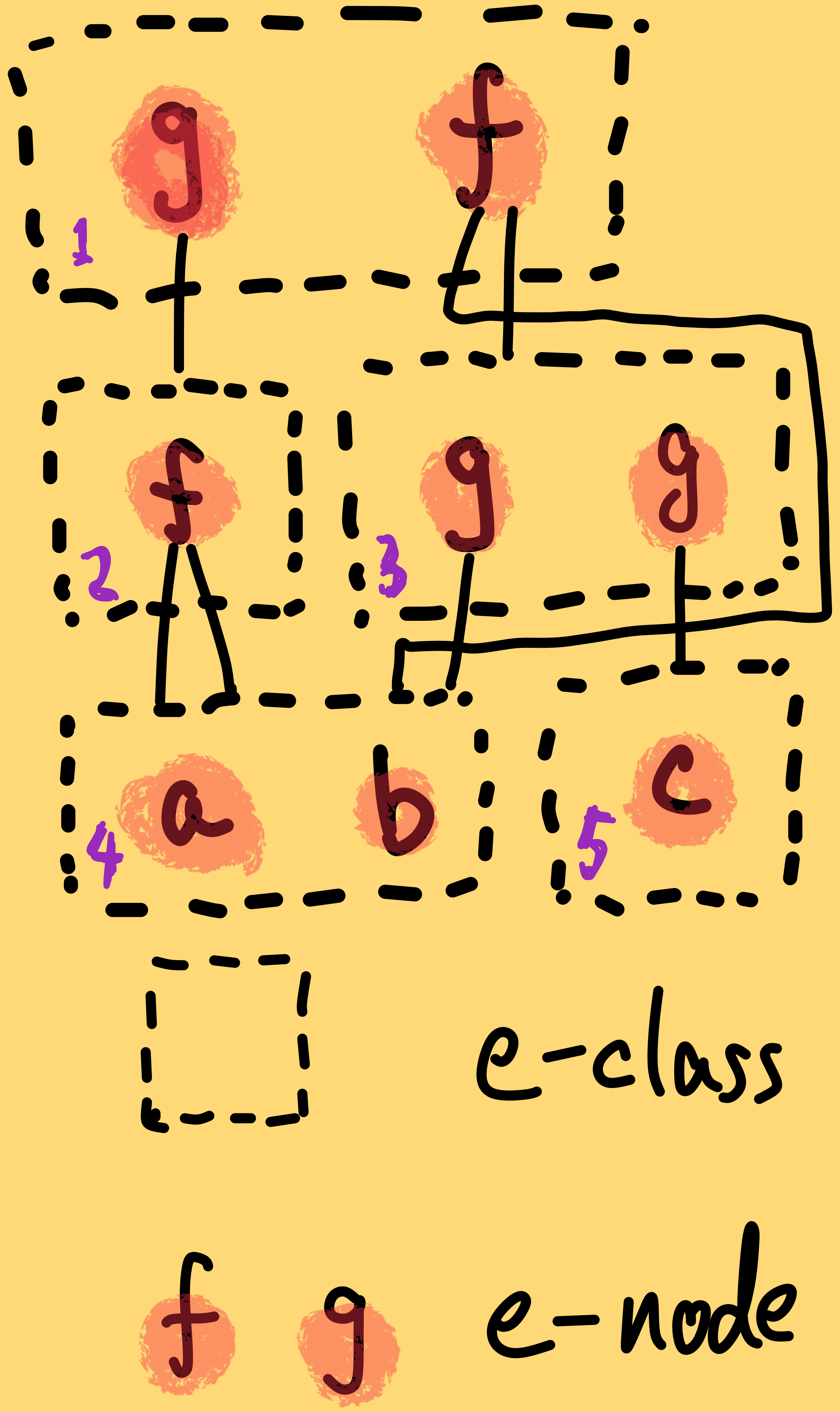
E-graphs



e-class

e-node

E-graphs



e-class 1 represents

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

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

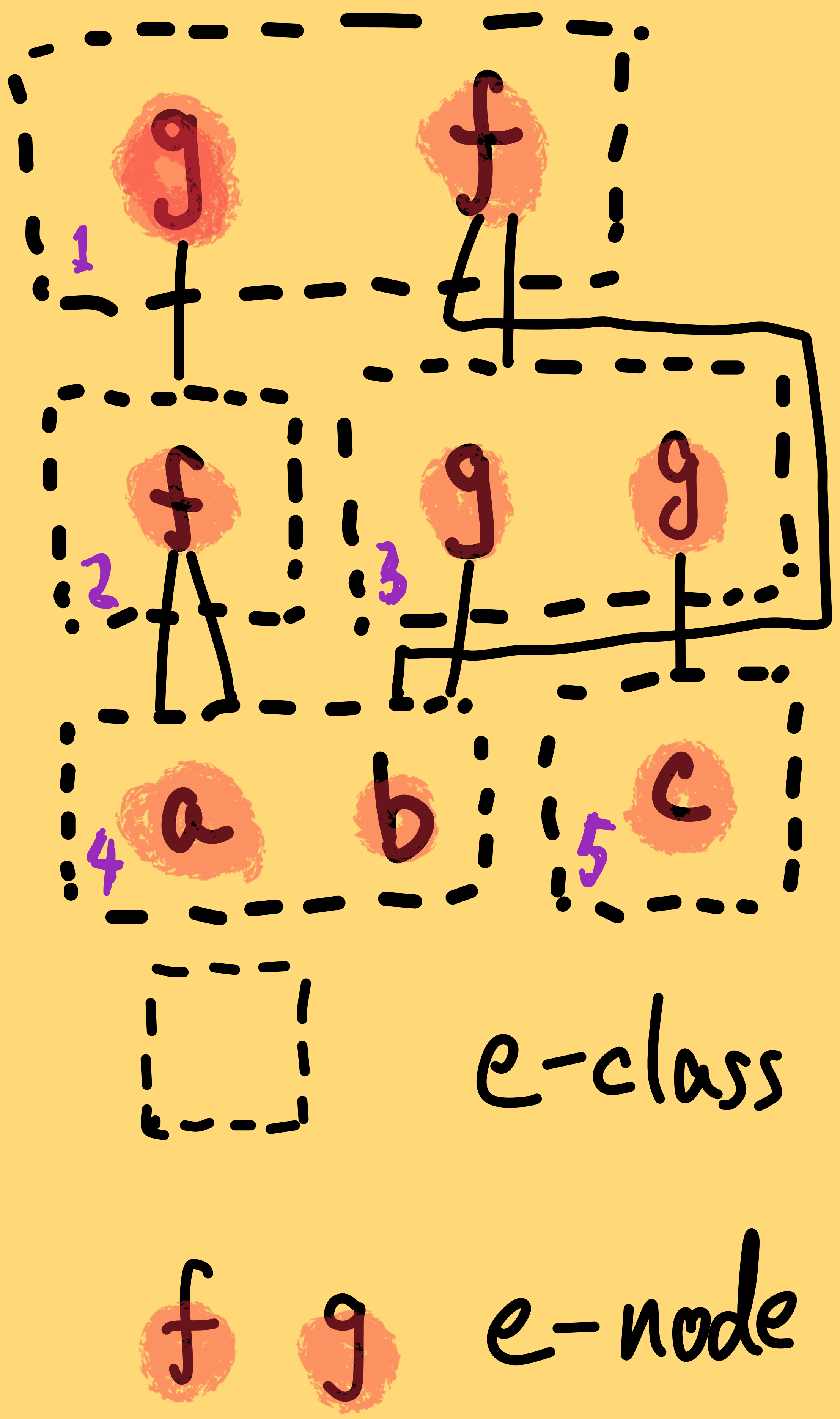
$g(f(b,a))$ $f(b,g(a))$

$g(f(b,b))$ $f(b,g(b))$

$f(a,g(c))$

$f(b,g(c))$

E-graphs



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

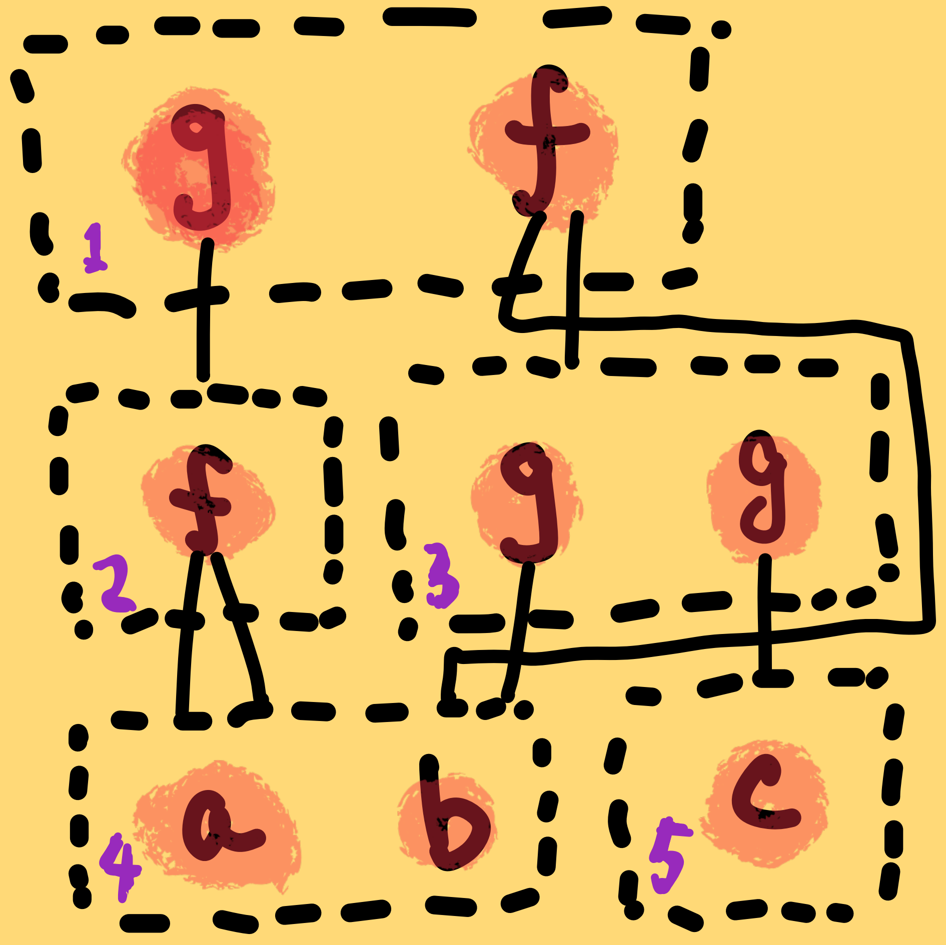
- E-matching is the query that finds patterns in an e-graph.

E-matching

- E-matching is the query that finds patterns in an e-graph.
- More formally, it finds substitutions mapping variables to e-class such that the e-graph contains the instantiated terms.

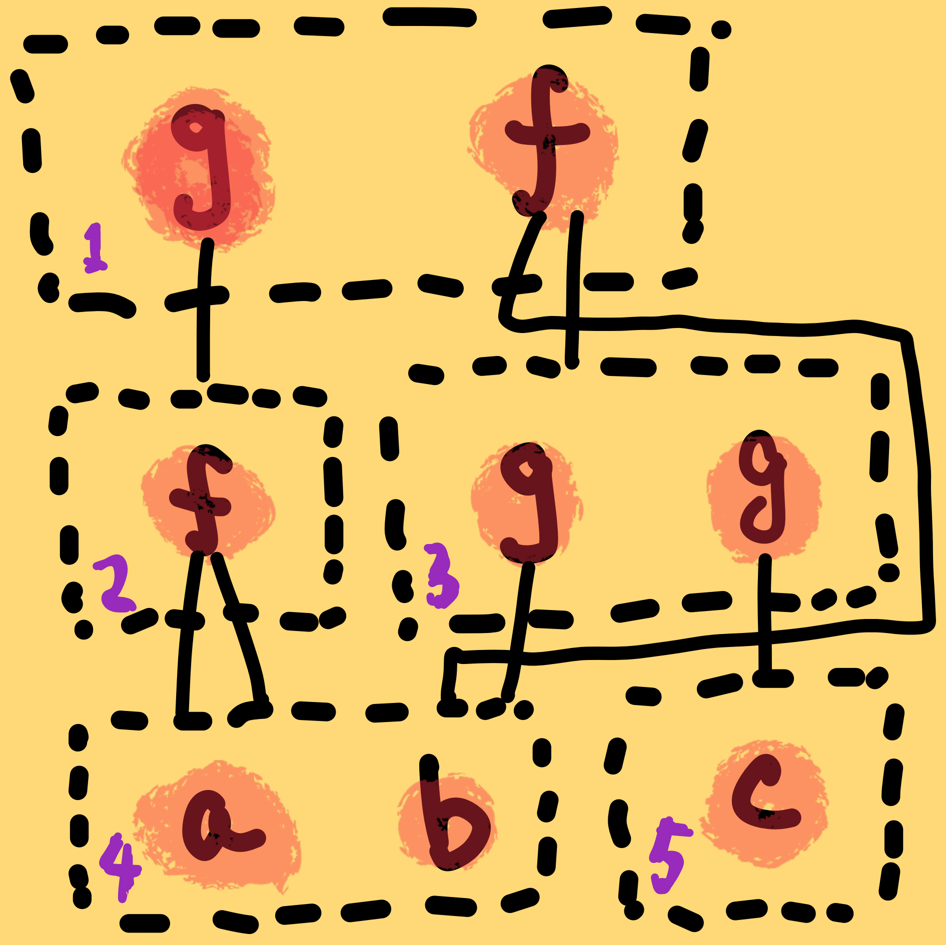
E-matching

- E-matching is the query that finds patterns in an e-graph.
- More formally, it finds substitutions mapping variables to e-class such that the e-graph contains the instantiated terms.



E-matching

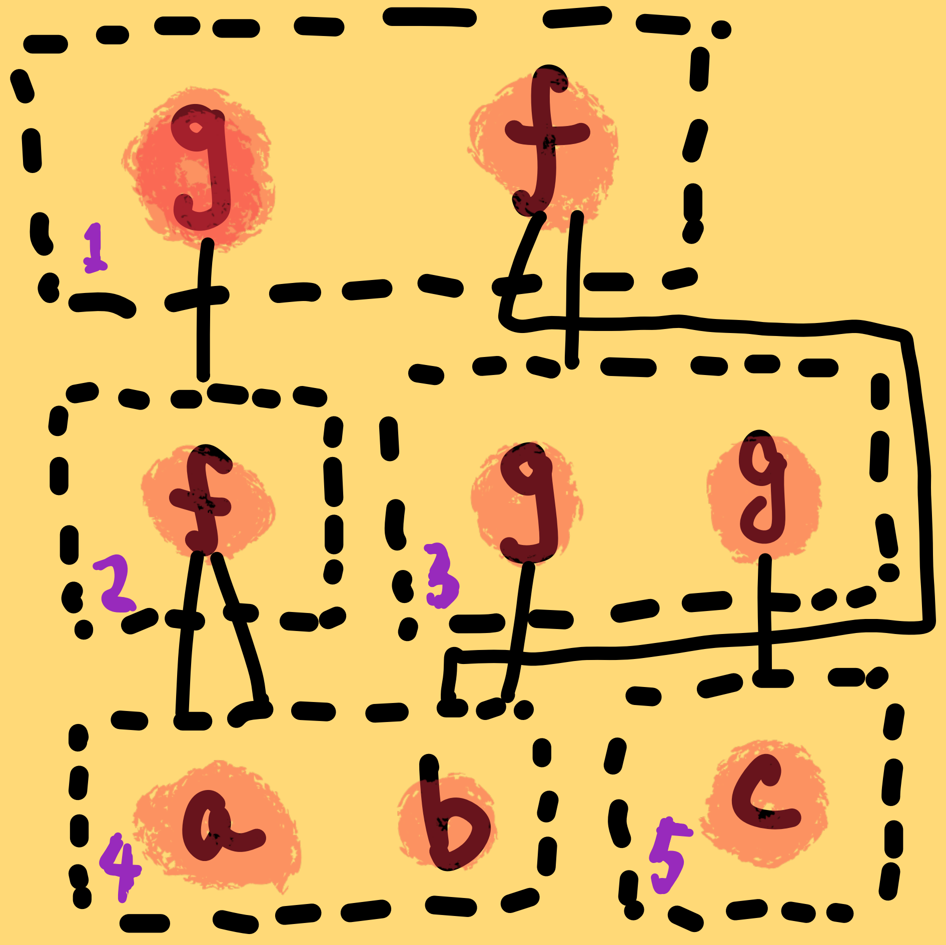
- E-matching is the query that finds patterns in an e-graph.
- More formally, it finds substitutions mapping variables to e-class such that the e-graph contains the instantiated terms.



pattern $f(\alpha, g(\alpha))$ corresponds to

E-matching

- E-matching is the query that finds patterns in an e-graph.
- More formally, it finds substitutions mapping variables to e-class such that the e-graph contains the instantiated terms.



pattern $f(\alpha, g(\alpha))$ corresponds to

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

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

all witnessed by subst. $\{\alpha \mapsto 4\}$

E-matching

- E-matching is the query that finds patterns in an e-graph.
- More formally, it finds substitutions mapping variables to e-class such that the e-graph contains the instantiated terms.

E-matching

- E-matching is the query that finds patterns in an e-graph.
- More formally, it finds substitutions mapping variables to e-class such that the e-graph contains the instantiated terms.
- E-matching is NP-complete w.r.t. the pattern + e-graph size.

E-matching

- E-matching is the query that finds patterns in an e-graph.
- More formally, it finds substitutions mapping variables to e-class such that the e-graph contains the instantiated terms.
- E-matching is NP-complete w.r.t. the pattern + e-graph size.
- Responsible for 60-90% of the run time.

E-matching

- E-matching is the query that finds patterns in an e-graph.
- More formally, it finds substitutions mapping variables to e-class such that the e-graph contains the instantiated terms.
- E-matching is NP-complete w.r.t. the pattern + e-graph size.
- Responsible for 60-90% of the run time.

Bottleneck!

Existing e-matching algorithms

Existing e-matching algorithms

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

Existing e-matching algorithms

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

Existing e-matching algorithms

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

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

Existing e-matching algorithms

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

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

Existing e-matching algorithms

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

for all e-class c in e-graph E:
  for all f node n_1 in c:
    subst = $\{\alpha \mapsto n_1.\text{child}_1, \text{root} \mapsto c\}$
    for all g node n_2 in $n_1.\text{child}_2$:
      if subst[α] = $n_2.\text{child}_1$:
        yield subst

Existing e-matching algorithms

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

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

$\underline{\underline{O(n^2)}}$

Existing e-matching algorithms

Existing e-matching algorithms

- Relies on naïve backtracking.

Existing e-matching algorithms

- Relies on naïve backtracking.
- Uses several ad hoc algorithms for different patterns.

Existing e-matching algorithms

- Relies on naïve backtracking.
- Uses several ad hoc algorithms for different patterns.
- Exploit only the structural constraints of a pattern, equality constraints are ignored.

Existing e-matching algorithms

- Relies on naïve backtracking.
- Uses several ad hoc algorithms for different patterns.
- Exploit only the structural constraints of a pattern, equality constraints are ignored.
- No theoretical guarantees besides a trivial bound $O(n^k)$.

Existing e-matching algorithms

- Relies on naïve backtracking.
- Uses several ad hoc algorithms for different patterns.
- Exploit only the structural constraints of a pattern, equality constraints are ignored.
- No theoretical guarantees besides a trivial bound $O(n^k)$.

We can do better!
(despite the NP-completeness)

Observations

Observations

E-matching

Finding substitutions such that the e-graph contains terms from the instantiated patterns.

Observations

E-matching

Finding substitutions such that the e-graph contains terms from the instantiated patterns.

Conjuctive query

Finding substitutions such that the relational database contains all the instantiated atoms of the query.

Observations

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

E-matching

Finding substitutions such that the e-graph contains terms from the instantiated patterns.

Conjuctive query

Finding substitutions such that the relational database contains all the instantiated atoms of the query.

Observations

Relational queries only involving joins

e.g. $Q(a,c) \text{ :- } R(a,b), S(b,c)$

E-matching

Finding substitutions such that the e-graph contains terms from the instantiated patterns.

Conjuctive query

Finding substitutions such that the relational database contains all the instantiated atoms of the query.

Observations

Relational queries only involving joins

e.g. $Q(a,c) :- R(a,b), S(b,c)$

E-matching

Finding substitutions such that the e-graph contains terms from the instantiated patterns.

Conjuctive query

Finding substitutions such that the relational database contains all the instantiated atoms of the query.

Observations

Relational queries only involving joins

e.g. Q(a,c) :- R(a,b), S(b,c)

E-matching

Finding substitutions such that the e-graph contains terms from the instantiated patterns.

Conjuctive query

Finding substitutions such that the relational database contains all the instantiated atoms of the query.

Observations

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

E-matching

Finding substitutions such that the e-graph contains terms from the instantiated patterns.

Conjuctive query

Finding substitutions such that the relational database contains all the instantiated atoms of the query.

Observations

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

E-matching

Finding substitutions such that the e-graph contains terms from the instantiated patterns.

Conjuctive query

Finding substitutions such that the relational database contains all the instantiated atoms of the query.

Observations

Relational queries only involving joins

e.g. $Q(a,c) :- R(a,b), S(b,c)$

E-matching

Finding substitutions such that the e-graph contains terms from the instantiated patterns.

Conjuctive query

Finding substitutions such that the relational database contains all the instantiated atoms of the query.

Observations

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

E-matching

Finding substitutions such that the e-graph contains terms from the instantiated patterns.

Conjuctive query

Finding substitutions such that the relational database contains all the instantiated atoms of the query.

Relational e-matching

Relational e-matching

$$\subseteq$$

Relational e-matching

E-graphs $\subseteq$

Relational e-matching

E-graphs $\subseteq$ Relational databases

Relational e-matching

E-graphs $\subseteq$ Relational databases

E-matching

Relational e-matching

E-graphs	$\subseteq$	Relational databases
E-matching		Conjunctive queries

Relational e-matching

E-graphs $\subseteq$ Relational databases

E-matching $\subseteq$ Conjunctive queries

Simpler!!

Relational e-matching

Relational e-matching

- Takes an e-graph and a list of patterns.

Relational e-matching

- Takes an e-graph and a list of patterns.
- Transform the e-graph to a relational database.

Relational e-matching

- Takes an e-graph and a list of patterns.
- Transform the e-graph to a relational database.
- Compiles all e-matching patterns to conjuctive queries.

Relational e-matching

- Takes an e-graph and a list of patterns.
- Transform the e-graph to a relational database.
- Compiles all e-matching patterns to conjuctive queries.
- Run the conjuctive queries on the relational database!

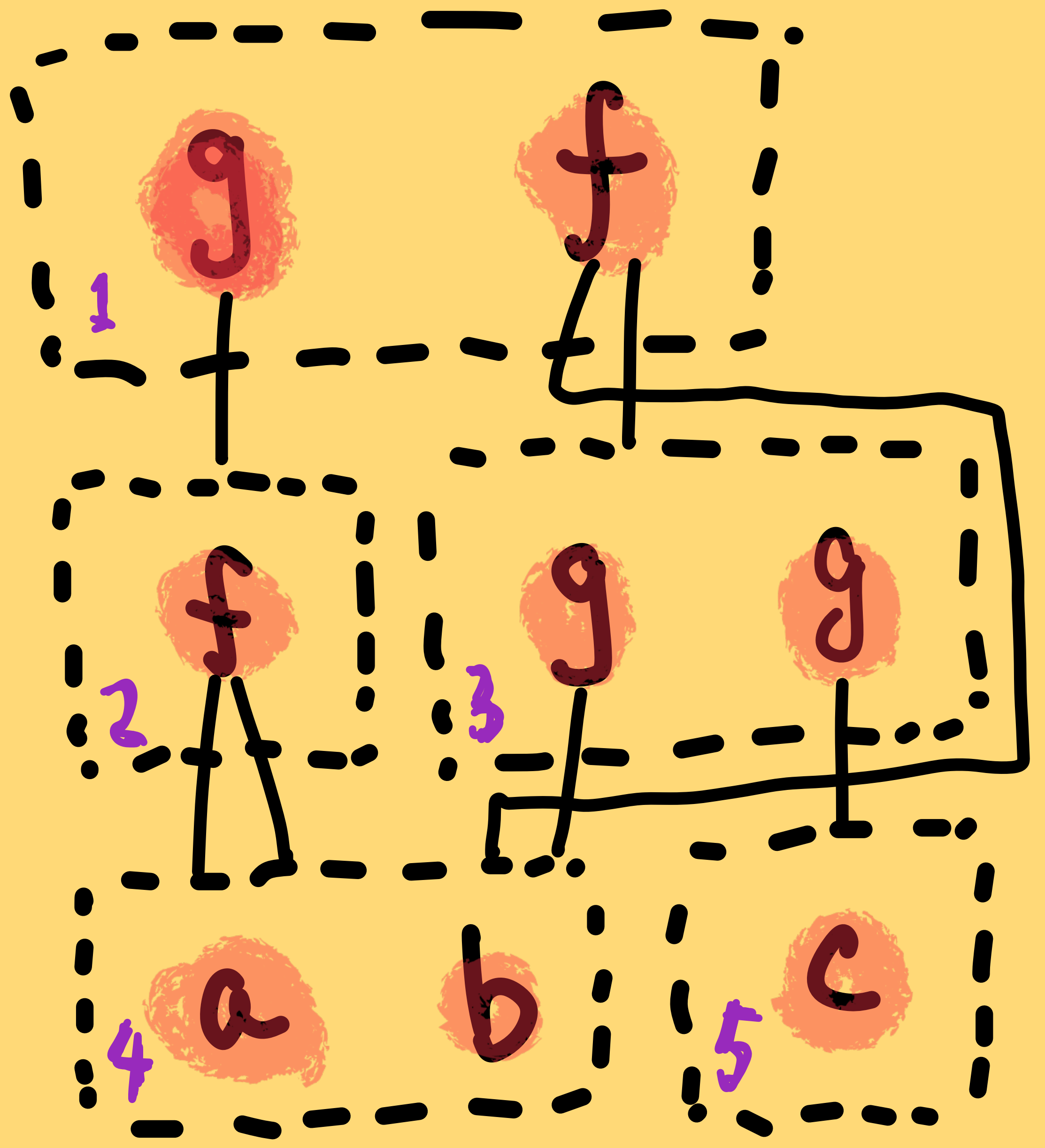
Relational e-matching

well suited for equality saturation

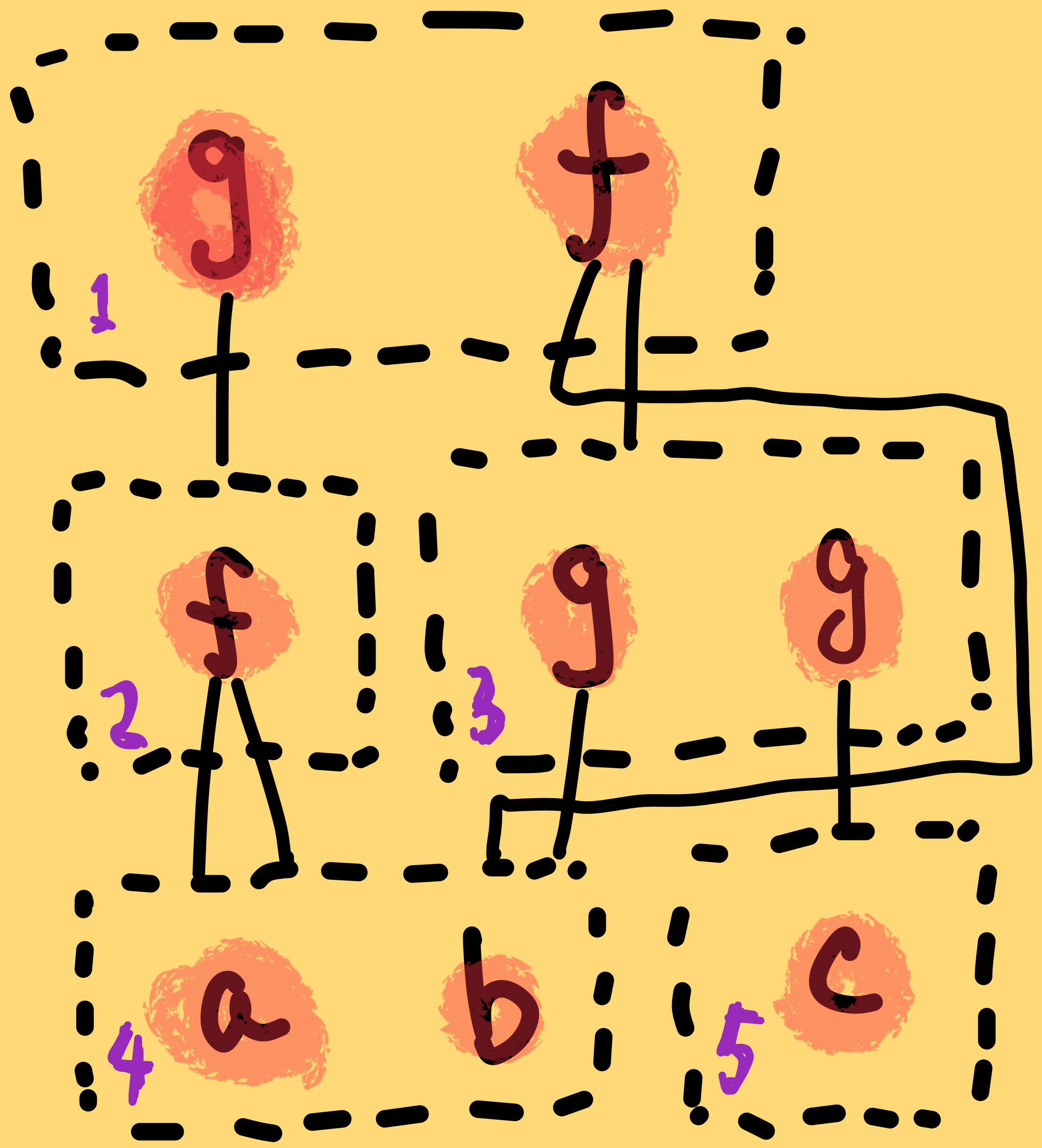
- Takes an e-graph and a list of patterns.
- Transform the e-graph to a relational database.
- Compiles all e-matching patterns to conjunctive queries.
- Run the conjunctive queries on the relational database!

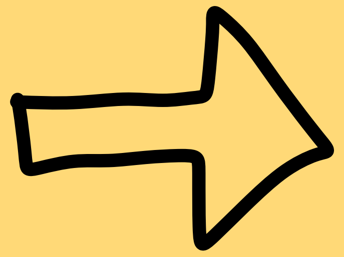
E-graphs $\rightarrow$ Relational database

E-graphs → Relational database

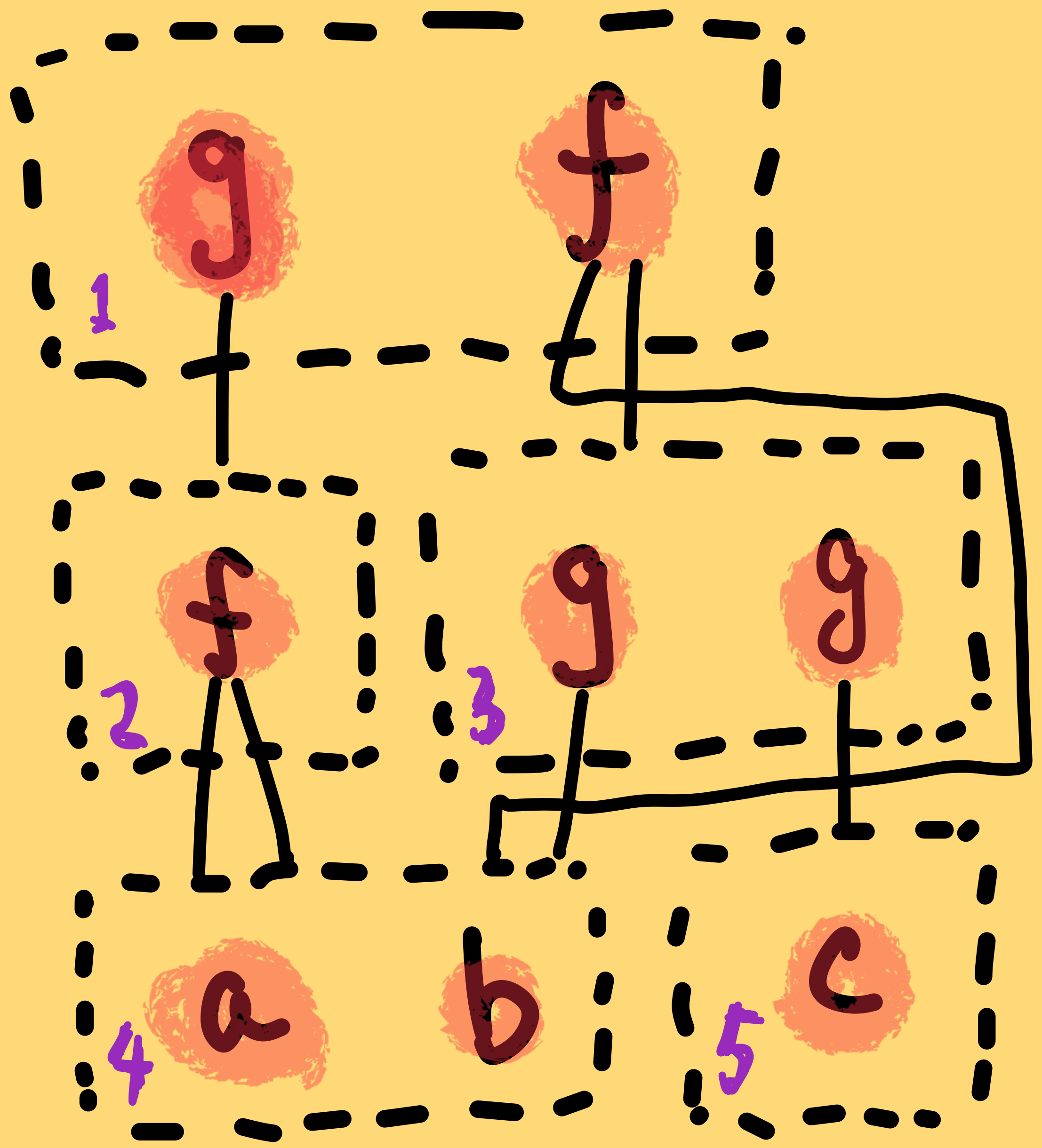


E-graphs → Relational database





E-graphs → Relational database



R_f

eclass-id	$child_1$	$child_2$

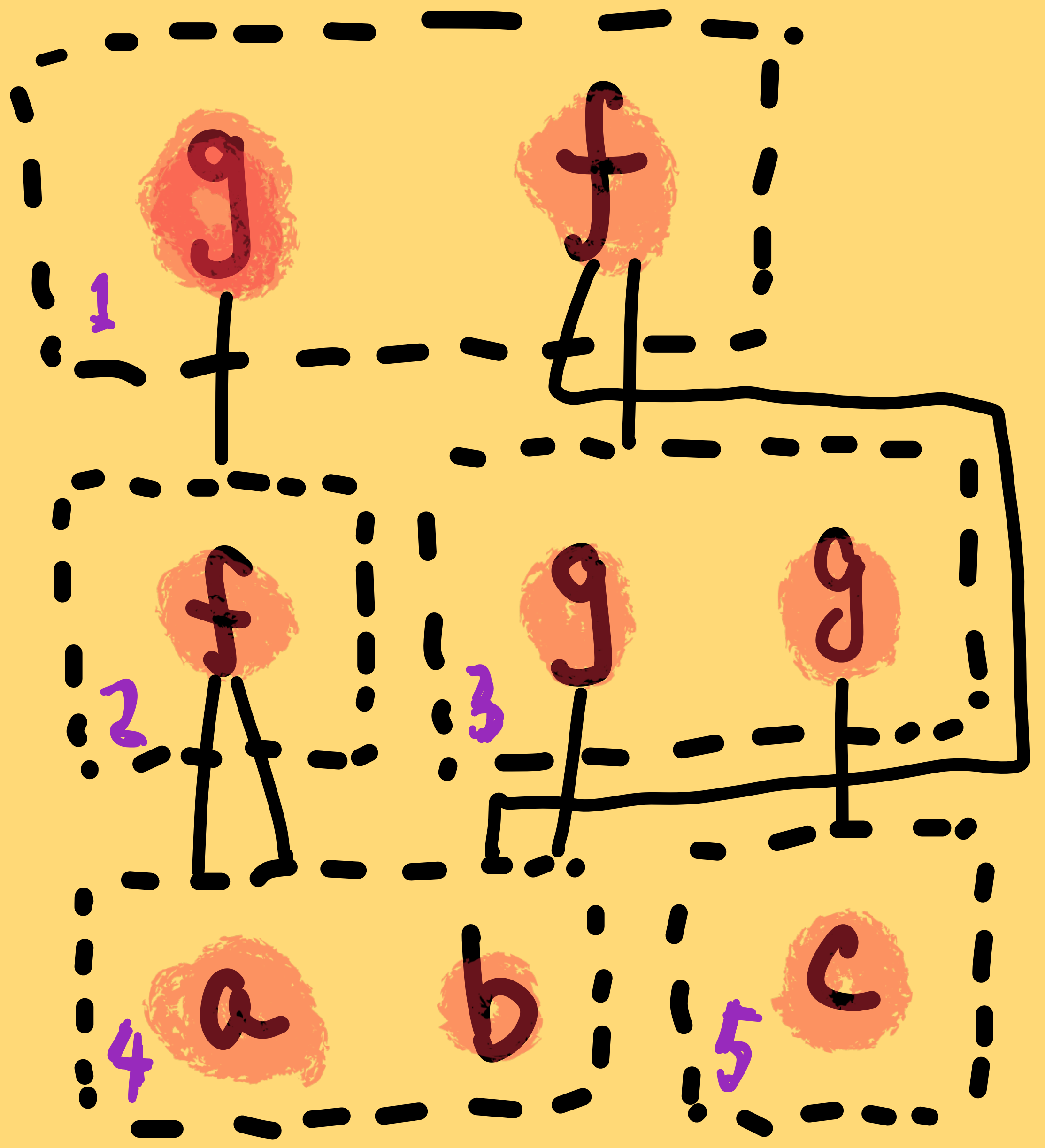
schema for function symbol f

R_g

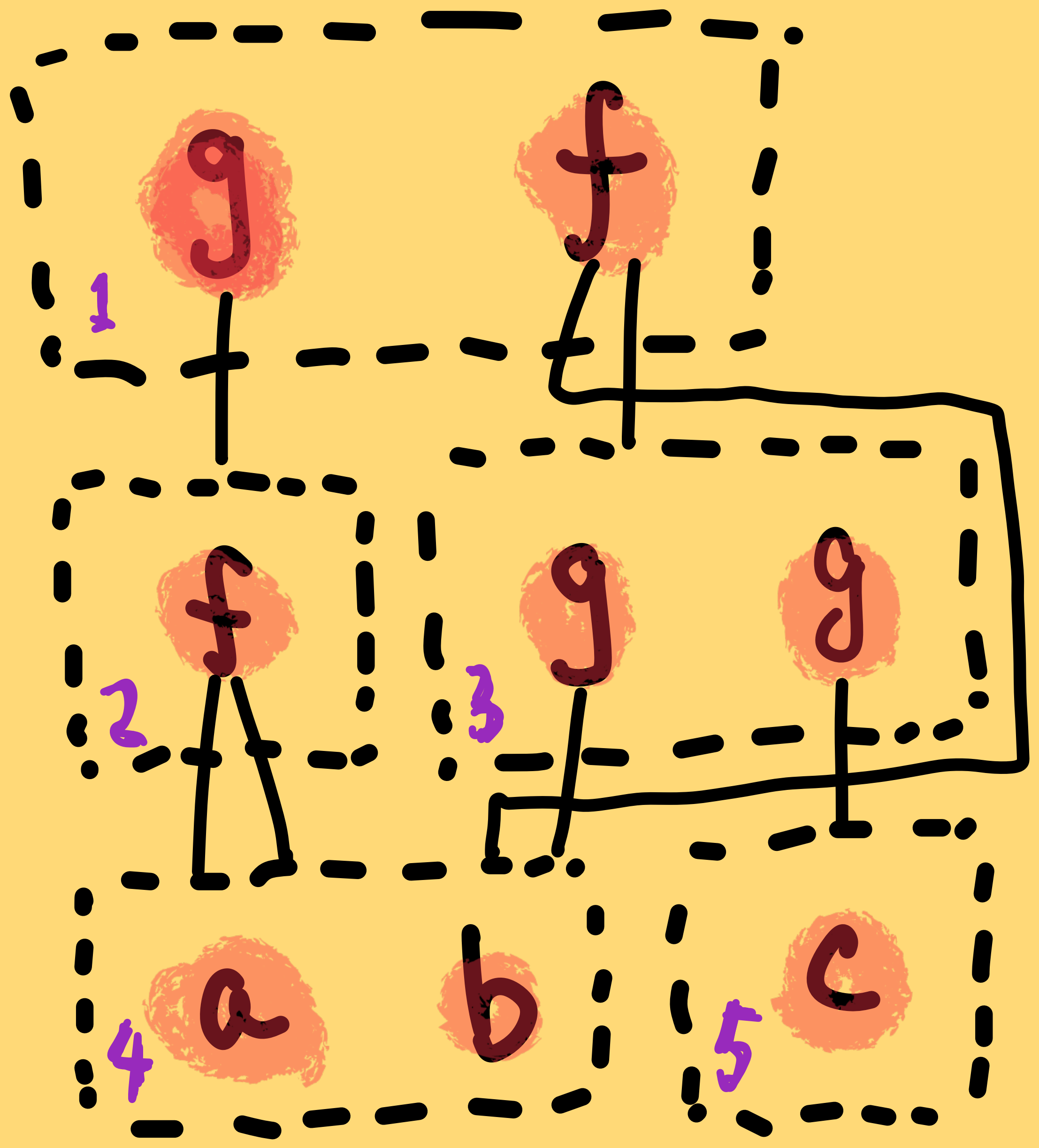
eclass-id	$child_1$

schema for function symbol g

E-graphs → Relational database



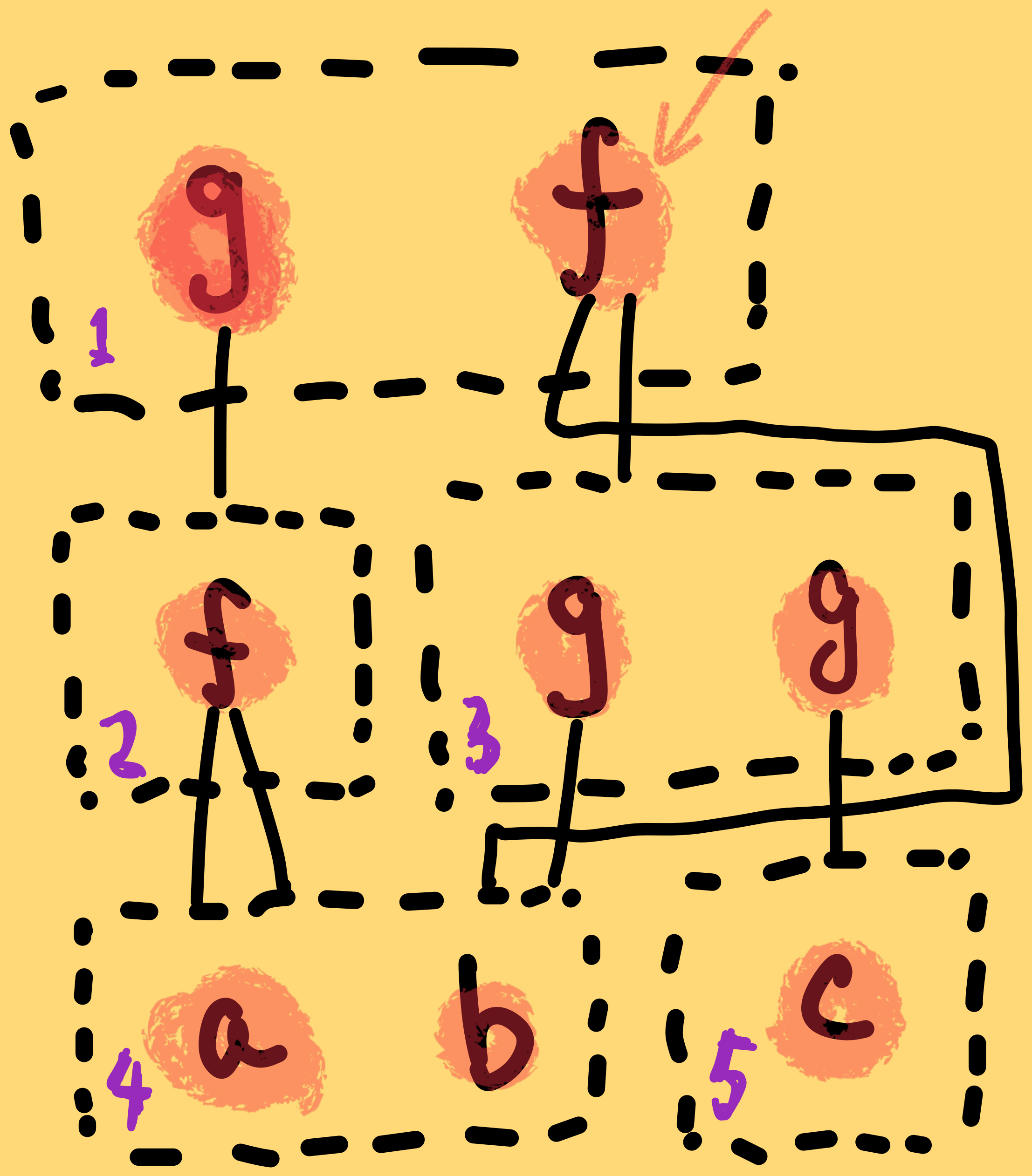
E-graphs → Relational database



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

eclass-id	$child_1$
1	2
3	4
3	5

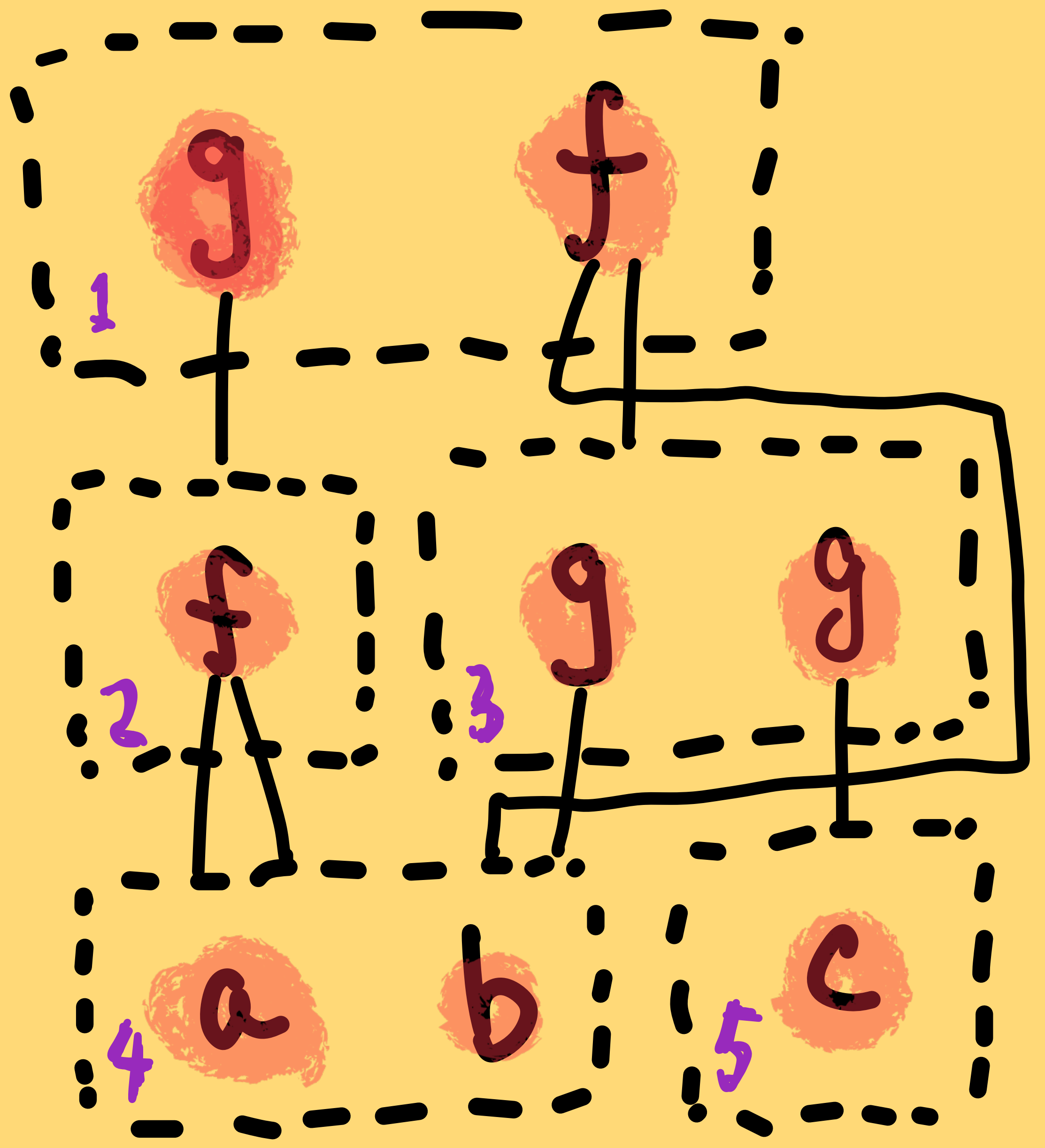
E-graphs → Relational database



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

eclass-id	$child_1$
1	2
3	4
3	5

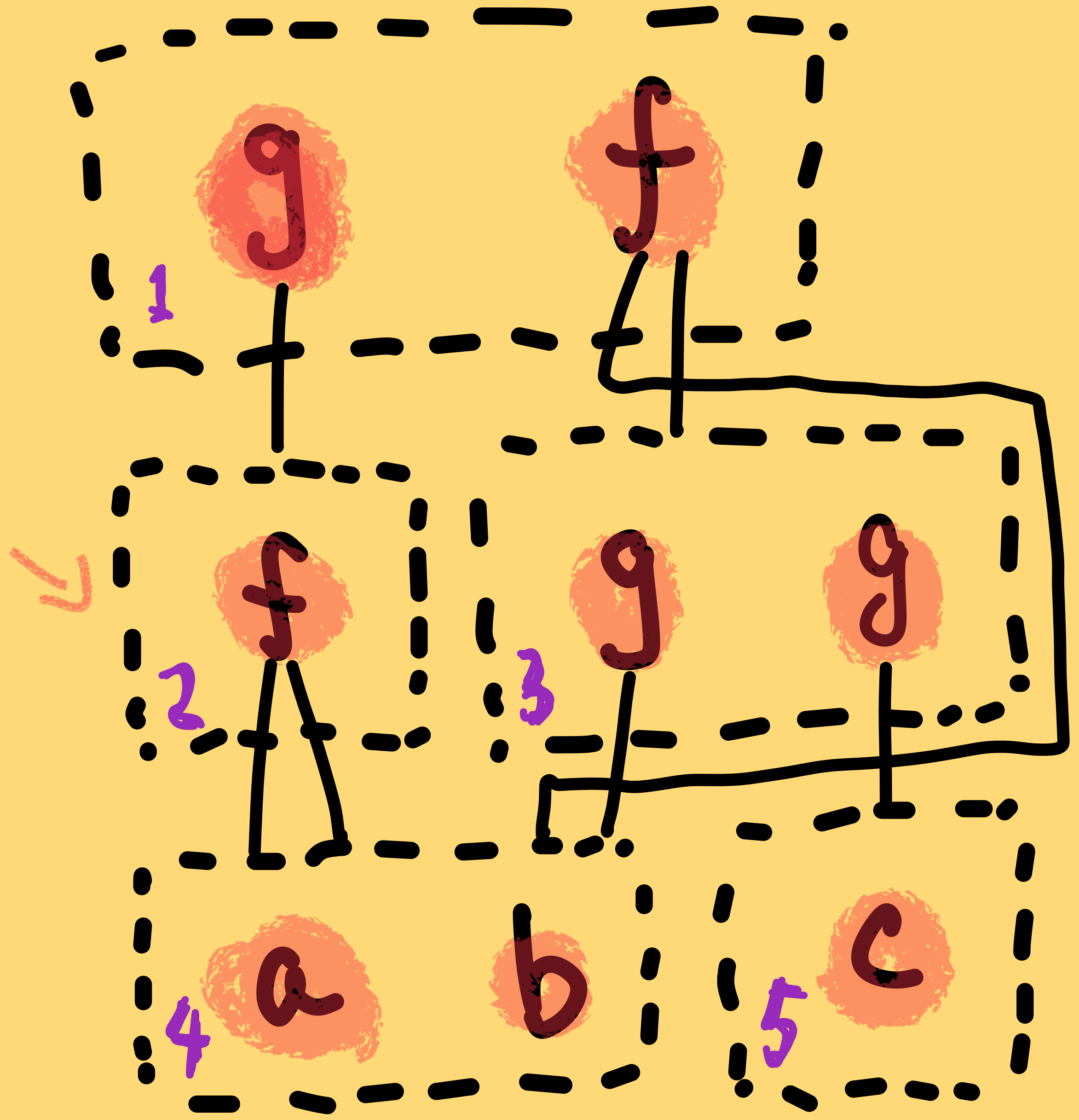
E-graphs → Relational database



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

eclass-id	$child_1$
1	2
3	4
3	5

E-graphs → Relational database



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

eclass-id	$child_1$
1	2
3	4
3	5

E-matching $\longrightarrow$ Conjuctive queries

E-matching $\longrightarrow$ Conjuctive queries

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

E-matching $\longrightarrow$ Conjuctive queries

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

E-matching $\longrightarrow$ Conjuctive queries

$f(\alpha, g(\alpha)) \quad \Longrightarrow \quad Q(root, \alpha) \text{ :- } R_f(root, \alpha, x), R_g(x, \alpha)$

E-matching ⟶ Conjuctive queries

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

$Q(root, \alpha) :-$
$R_f(root, \alpha, x), R_g(x, \alpha)$

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

⇒

E-matching ⟶ Conjuctive queries

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

$Q(root, \alpha) \mathrel{:-}$
$R_f(root, \alpha, x), R_g(x, \alpha)$

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

⇒

```
for all rf in Rf:
  if (rf.child2, rf.child1) in Rg:
    yield {root ↦ Rf.eclass-id,
           α ↦ Rf.child1}
```

E-matching $\longrightarrow$ Conjuctive queries

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

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

$\Rightarrow$

$Q(root, \alpha) :- R_f(root, \alpha, x), R_g(x, \alpha)$

```
for all rf in Rf:
  if (rf.child2, rf.child1) in Rg:
    yield {root ↦ Rf.eclass-id,
           α ↦ Rf.child1}
```

$O(n)$

E-matching ⟶ Conjuctive queries

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

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

$O(n^2)$

⇒

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

```
for all rf in Rf:
  if (rf.child2, rf.child1) in Rg:
    yield {root ↦ Rf.eclass-id,
           α ↦ Rf.child1}
```

$O(n)$

E-matching $\longrightarrow$ Conjuctive queries

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

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

$O(n^2)$

$\Rightarrow$

Faster!!

$Q(root, \alpha) :-$

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

```
for all rf in Rf:
  if (rf.child2, rf.child1) in Rg:
    yield {root ↦ Rf.eclass-id,
           α ↦ Rf.child1}
```

$O(n)$

Why faster?

Why faster?

$f(\alpha, g(\alpha))$: enumerate all terms of $f(\alpha, g(\beta))$, and check if $\alpha = \beta$ only before yielding.

Why faster?

$f(\alpha, g(\alpha))$: enumerate all terms of $f(\alpha, g(\beta))$, and check if $\alpha = \beta$ only before yielding.

$Q(root, \alpha)$:-
$Rf(root, \alpha, x), Rg(x, \alpha)$: build indices on both α and x and only enumerate terms where constraints on both α and x are satisfied.

Why faster?

$f(\alpha, g(\alpha))$: enumerate all terms of $f(\alpha, g(\beta))$, and check if $\alpha = \beta$ only before yielding.

$Q(\text{root}, \alpha)$:-
$R_f(\text{root}, \alpha, x), R_g(x, \alpha)$: build indices on both α and x and only enumerate terms where constraints on both α and x are satisfied.

equality constraint

structural constraint

Which CQ algorithm to use?

Which CQ algorithm to use?

Consider matching law of distributivity $(\alpha * \beta) + (\alpha * \gamma)$

Which CQ algorithm to use?

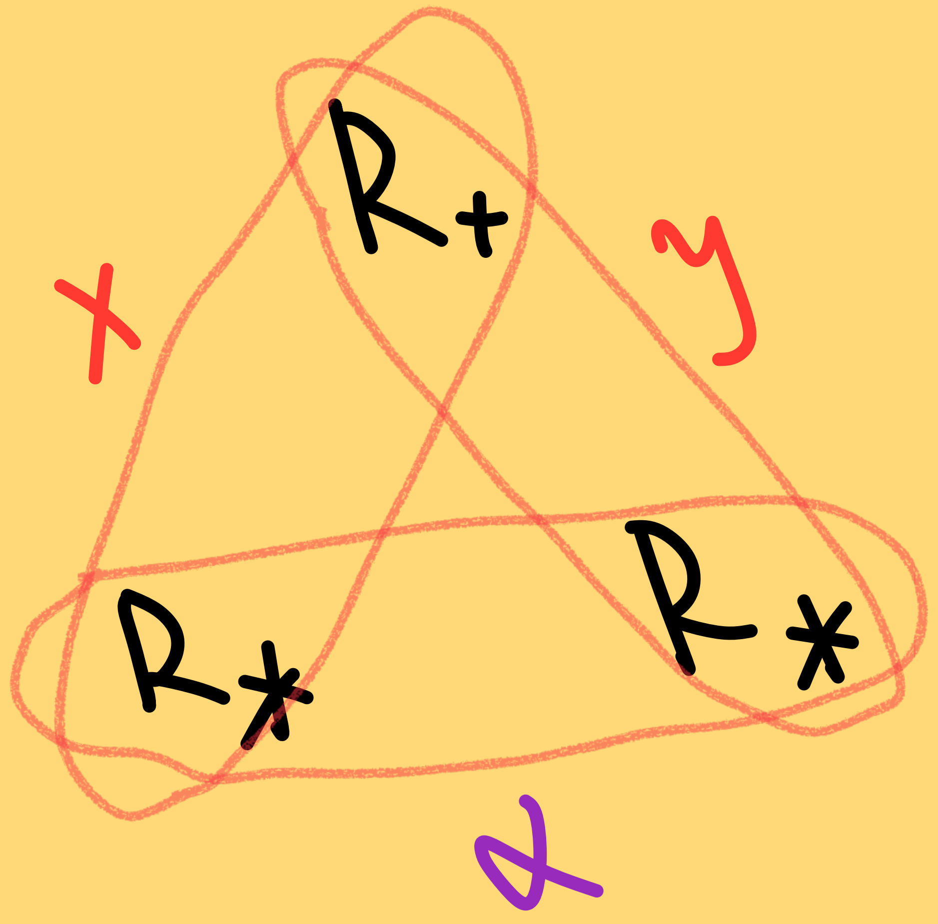
Consider matching law of distributivity $(\alpha * \beta) + (\alpha * \gamma)$

$$Q(root, \alpha, \beta, \gamma) :- R_+(root, x, y), R_*(x, \alpha, \beta), R_*(y, \alpha, \gamma)$$

Which CQ algorithm to use?

Consider matching law of distributivity $(\alpha * \beta) + (\alpha * \gamma)$

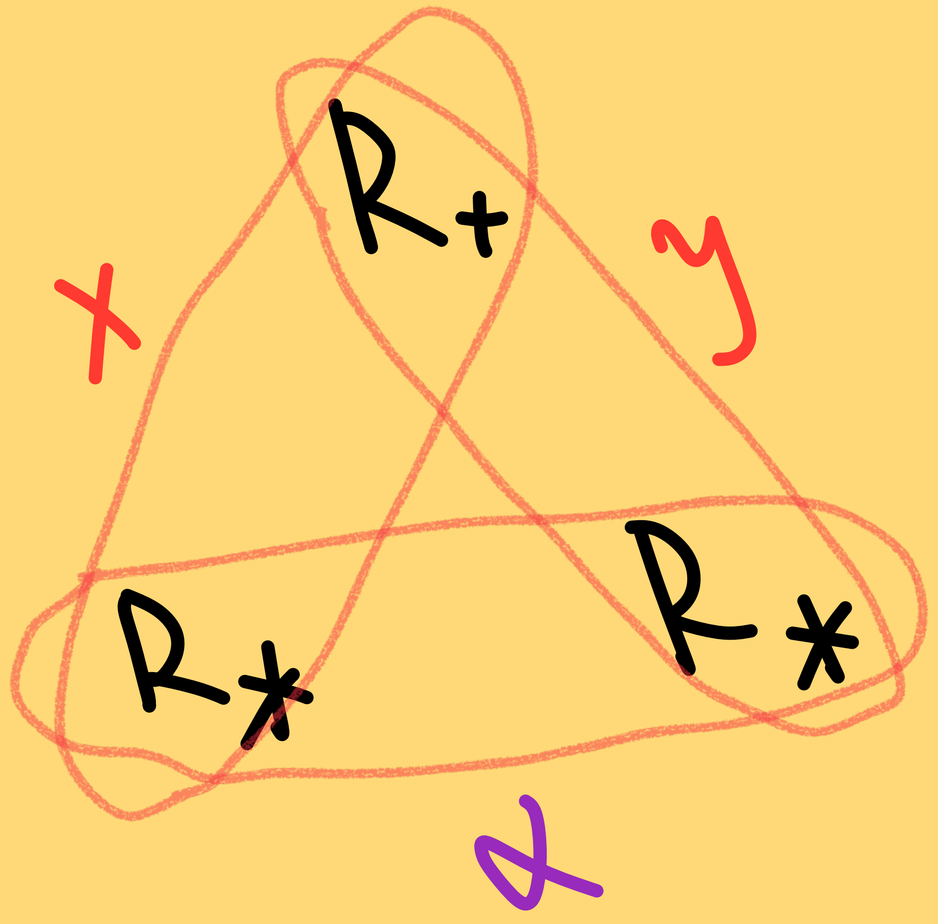
$$Q(root, \alpha, \beta, \gamma) :- R_+(root, x, y), R_*(x, \alpha, \beta), R_*(y, \alpha, \gamma)$$



Which CQ algorithm to use?

Consider matching law of distributivity $(\alpha * \beta) + (\alpha * \gamma)$

$$Q(root, \alpha, \beta, \gamma) :- R_+(root, x, y), R_*(x, \alpha, \beta), R_*(y, \alpha, \gamma)$$



All two-way join plans (e.g. hash join, sort-merge join) are known to be suboptimal on cyclic queries like this triangle query.

Generic join algorithms

Generic join algorithms

- Multi-way join that avoids unnecessarily large intermediate relations (even for cyclic queries!).

Generic join algorithms

- Multi-way join that avoids unnecessarily large intermediate relations (even for cyclic queries!).
- Worst-case optimal algorithms (The AGM bound).

Generic join algorithms

- Multi-way join that avoids unnecessarily large intermediate relations (even for cyclic queries!).
- Worst-case optimal algorithms (The AGM bound).
- Efficient for both cyclic and acyclic queries.

Generic join algorithms

Generic join algorithms

- The triangle query complies to

Generic join algorithms

- The triangle query complies to

$\textbf{for all } \alpha \textbf{ in } R_*.\mathrm{child}_1 \cap R_*.\mathrm{child}_1:$

$\quad \textbf{for all } \beta \textbf{ in } R_*[\mathrm{child}_1=\alpha].\mathrm{child}_2:$

$\quad\quad \textbf{for all } \gamma \textbf{ in } R_*[\mathrm{child}_1=\alpha].\mathrm{child}_2:$

$\quad\quad\quad \textbf{for all } x \textbf{ in } R_+.\mathrm{child}_1 \cap R_*[\mathrm{child}_1=\alpha, \mathrm{child}_2=\beta].\text{eclass-id}:$

$\quad\quad\quad\quad \textbf{for all } y \textbf{ in } R_+[\mathrm{child}_1=x].\mathrm{child}_2 \cap R_*[\mathrm{child}_1=\alpha, \mathrm{child}_2=\gamma].\text{eclass-id}:$

$\quad\quad\quad\quad\quad \textbf{for all } \mathrm{root} \textbf{ in } R_+[\mathrm{child}_1=x, \mathrm{child}_2=y]:$

$\quad\quad\quad\quad\quad\quad \textbf{yield } \{\mathrm{root}, \alpha, \beta, \gamma\}$

Generic join algorithms

- The triangle query complies to

```
for all α in R*.child1 ∩ R*.child1:
  for all β in R*[child1=α].child2:
    for all γ in R*[child1=α].child2:
      for all x in R+.child1 ∩ R*[child1=α, child2=β].eclass-id:
        for all y in R+[child1=x].child2 ∩ R*[child1=α, child2=γ].eclass-id:
          for all root in R+[child1=x, child2=y]:
            yield {root, α, β, γ}
```

Process queries variable-by-variable (vs relation-by-relation)

Generic join algorithms

Generic join algorithms

- Relational e-matching preserves the worst-case optimality of generic join.

Generic join algorithms

- Relational e-matching preserves the worst-case optimality of generic join.
- Fix a pattern p. Let $M(p, E)$ be the set of substitutions yielded by e-matching on e-graph E with size n, relational e-matching runs in time

$$\tilde{O}(\max_E(|M(p,E)|)).$$

Generic join algorithms

- Relational e-matching preserves the worst-case optimality of generic join.
- Fix a pattern p. Let $M(p,E)$ be the set of substitutions yielded by e-matching on e-graph E with size n, relational e-matching runs in time

$$\tilde{O}(\max_E(|M(p,E)|)).$$

Optimal !!

(Preliminary) Evaluations

(Preliminary) Evaluations

- Linear patterns: patterns without equality constraints

(Preliminary) Evaluations

- Linear patterns: patterns without equality constraints
- Cyclic patterns: patterns that generate cyclic queries

(Preliminary) Evaluations

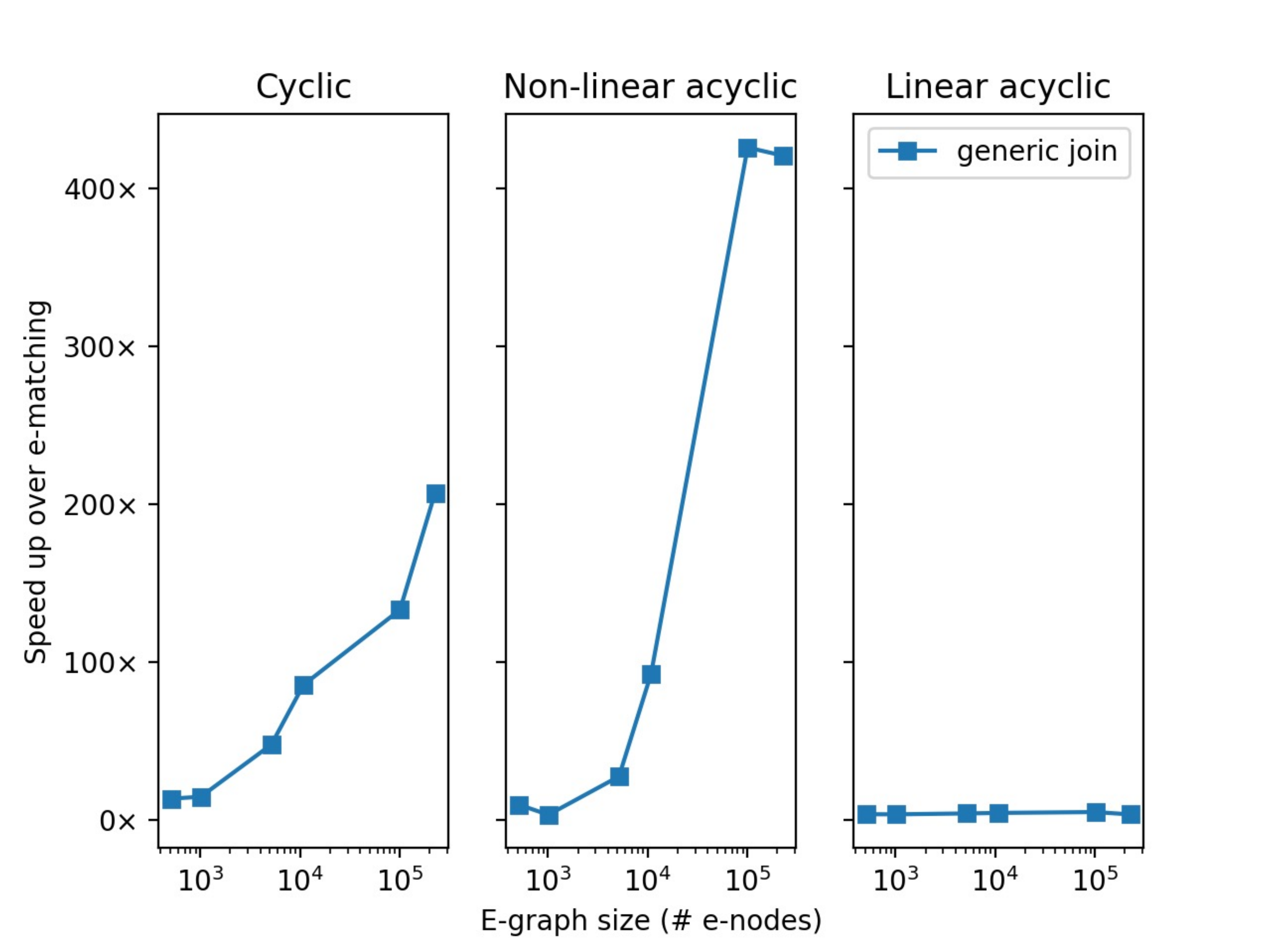
- Linear patterns: patterns without equality constraints
- Cyclic patterns: patterns that generate cyclic queries
- Acyclic nonlinear patterns: patterns with equality constraints but no cycles

(Preliminary) Evaluations

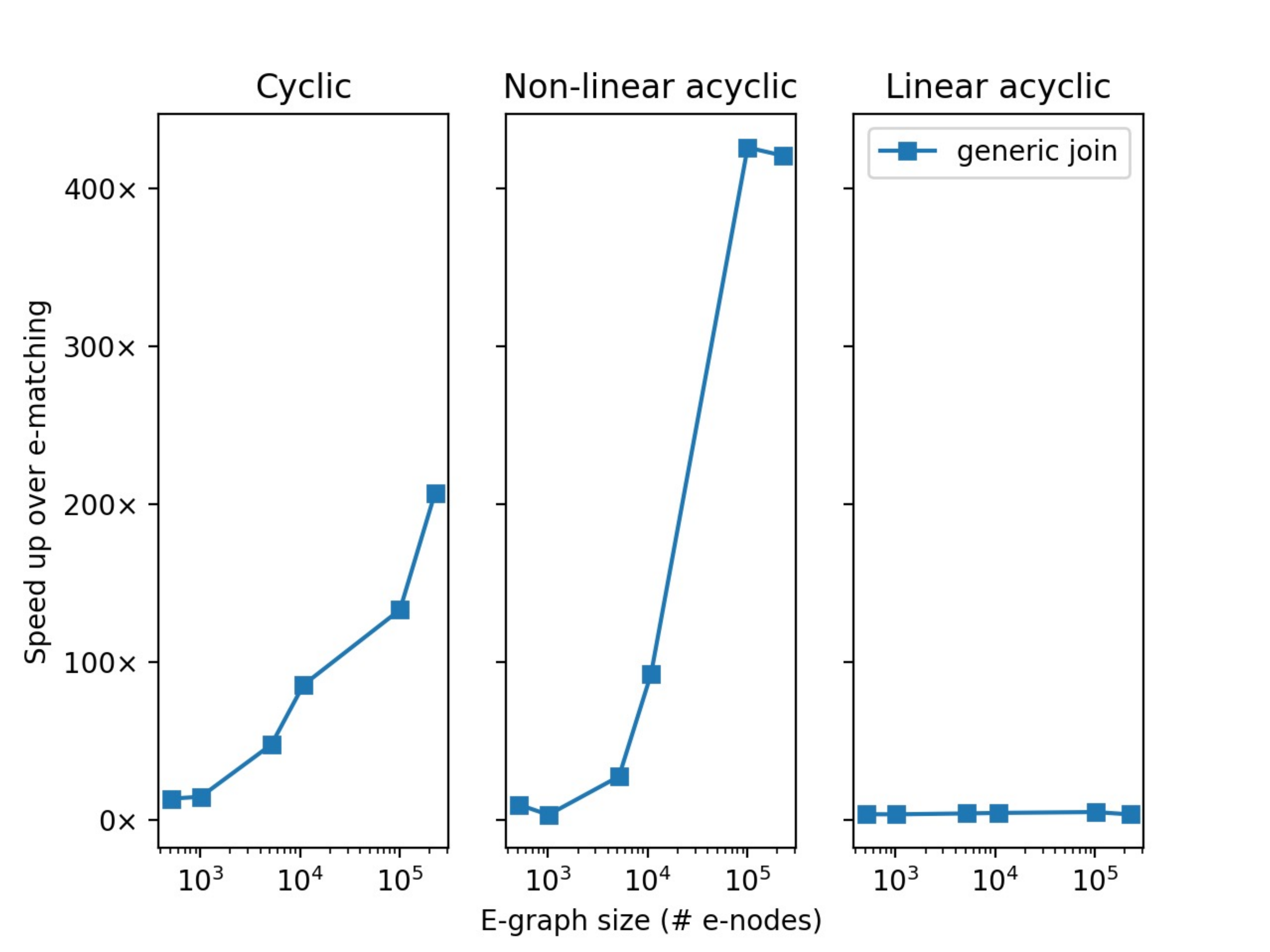
- Linear patterns: patterns without equality constraints
- Cyclic patterns: patterns that generate cyclic queries
- Acyclic nonlinear patterns: patterns with equality constraints but no cycles
- Singleton pattens: patterns with no nested subpatterns

(Preliminary) Evaluations

(Preliminary) Evaluations

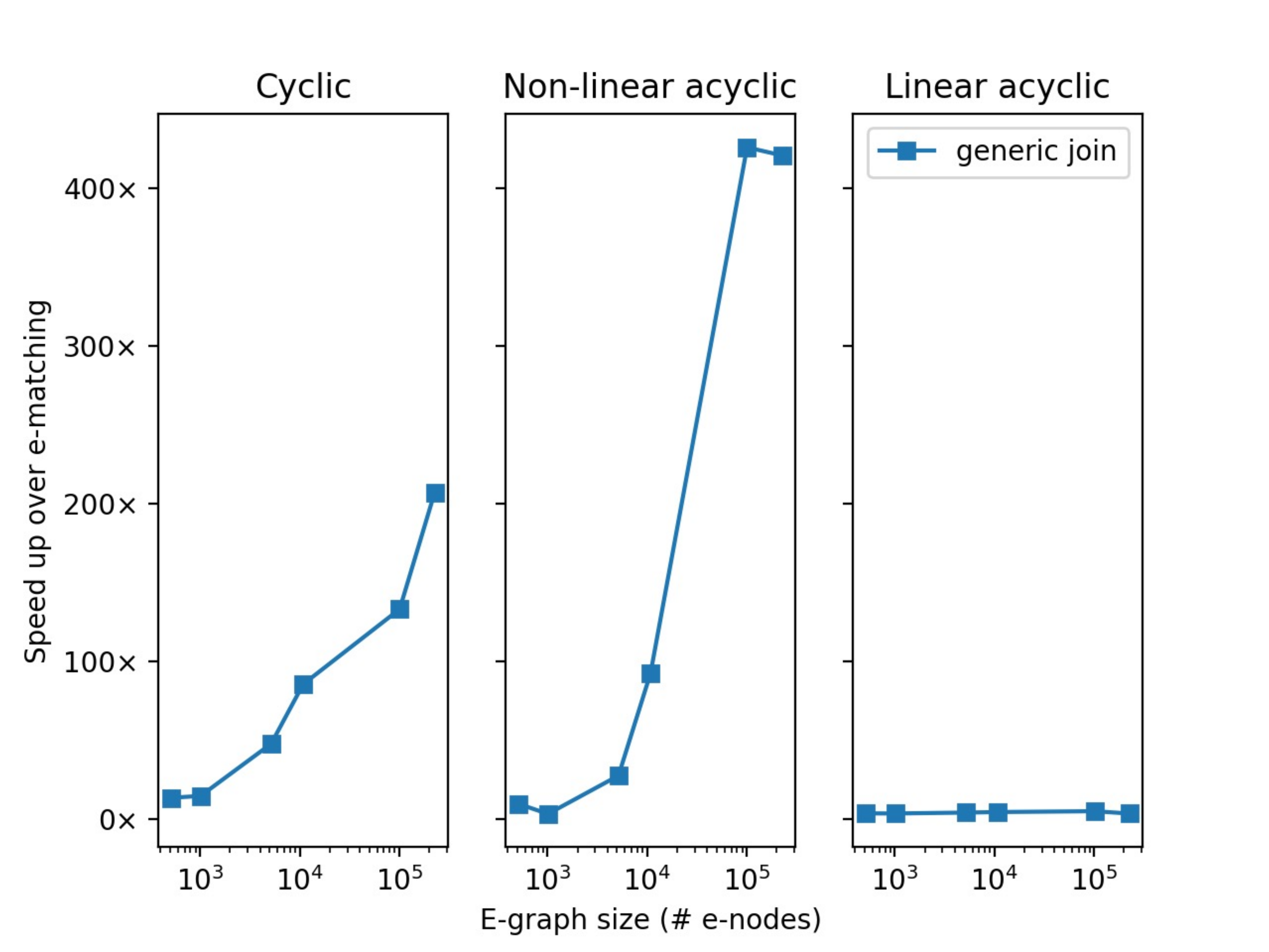


(Preliminary) Evaluations



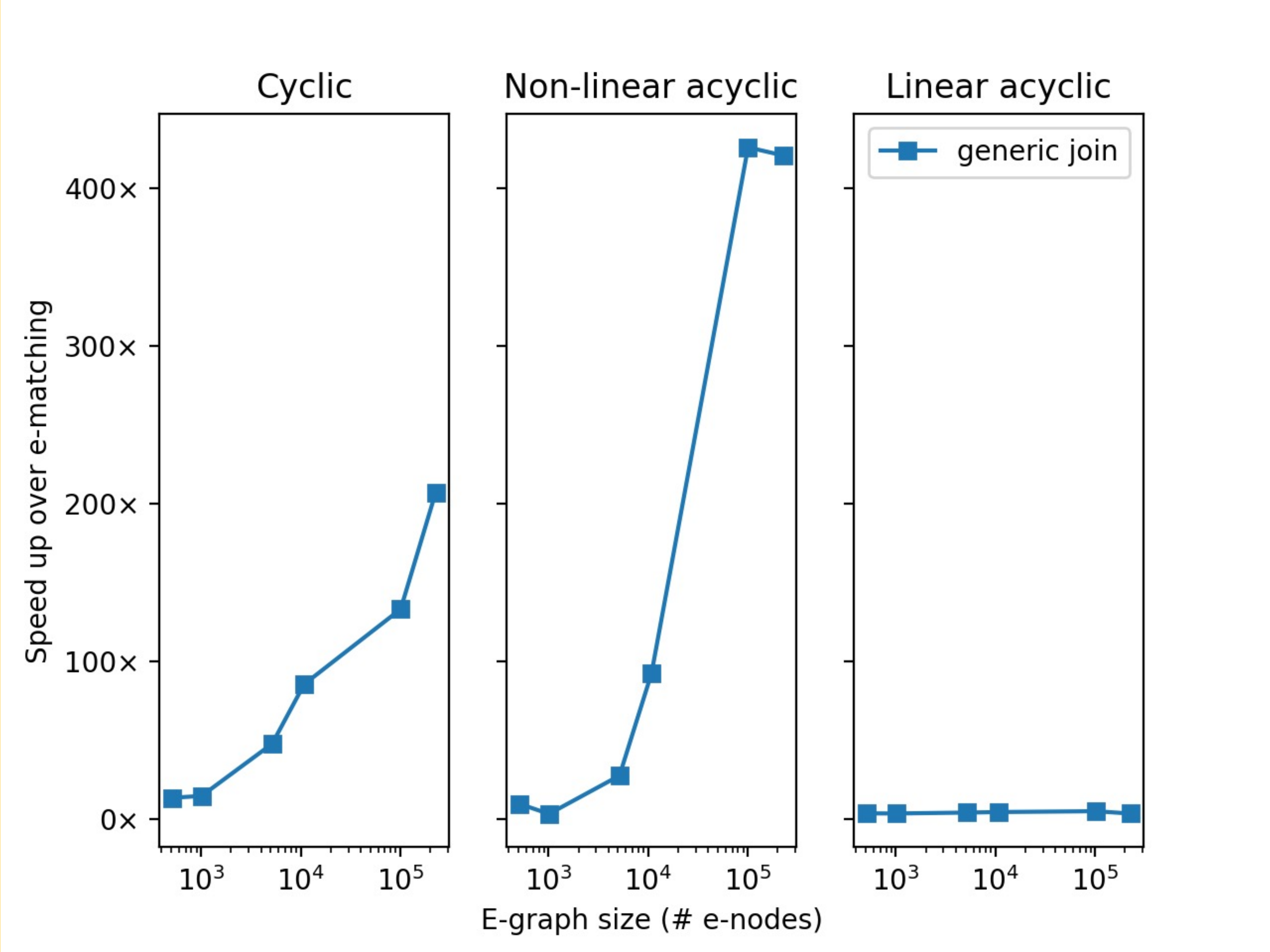
- Collected from egg's math test suite

(Preliminary) Evaluations



- Collected from egg's math test suite
- Asymptotic speedup on nonlinear patterns

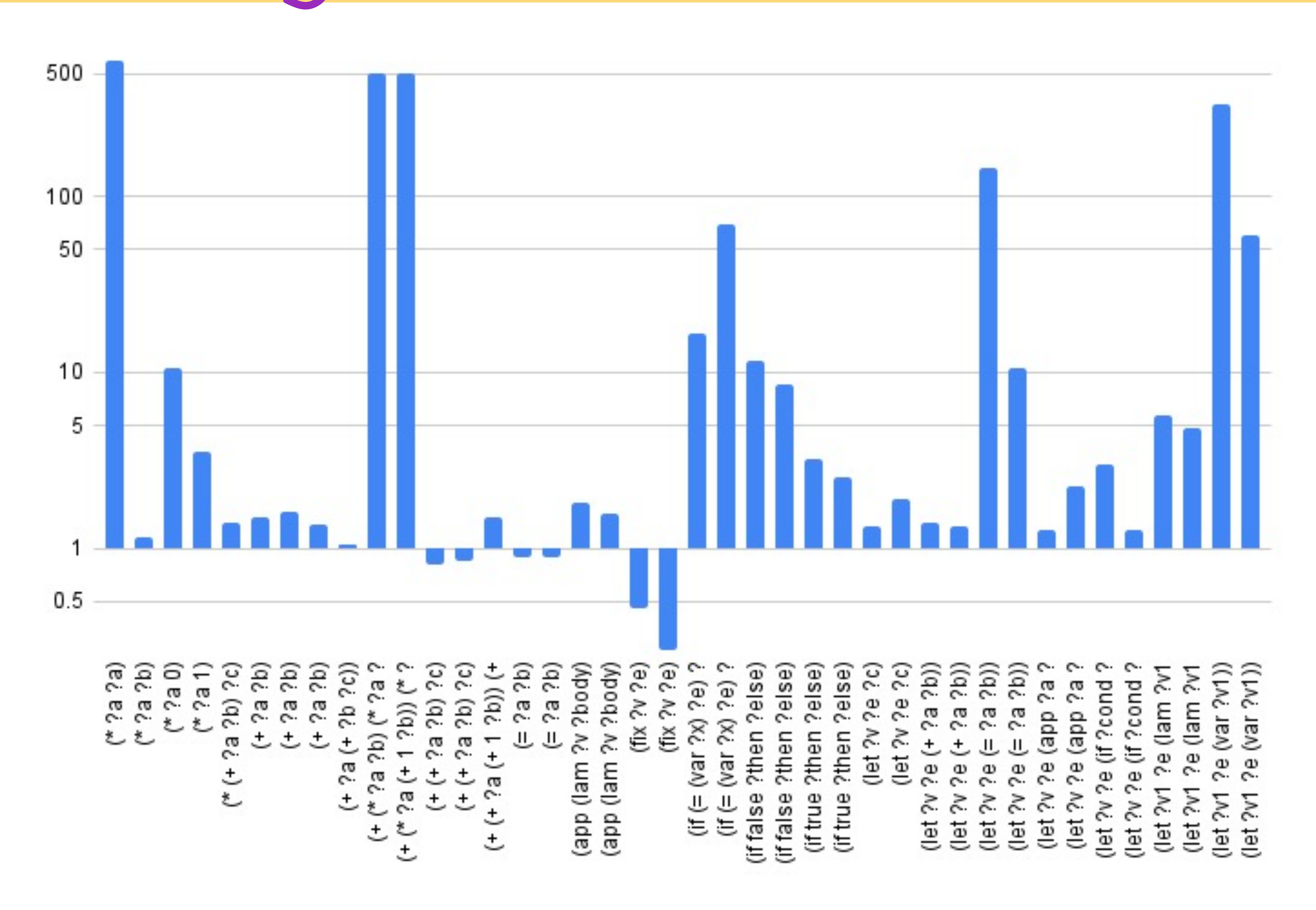
(Preliminary) Evaluations



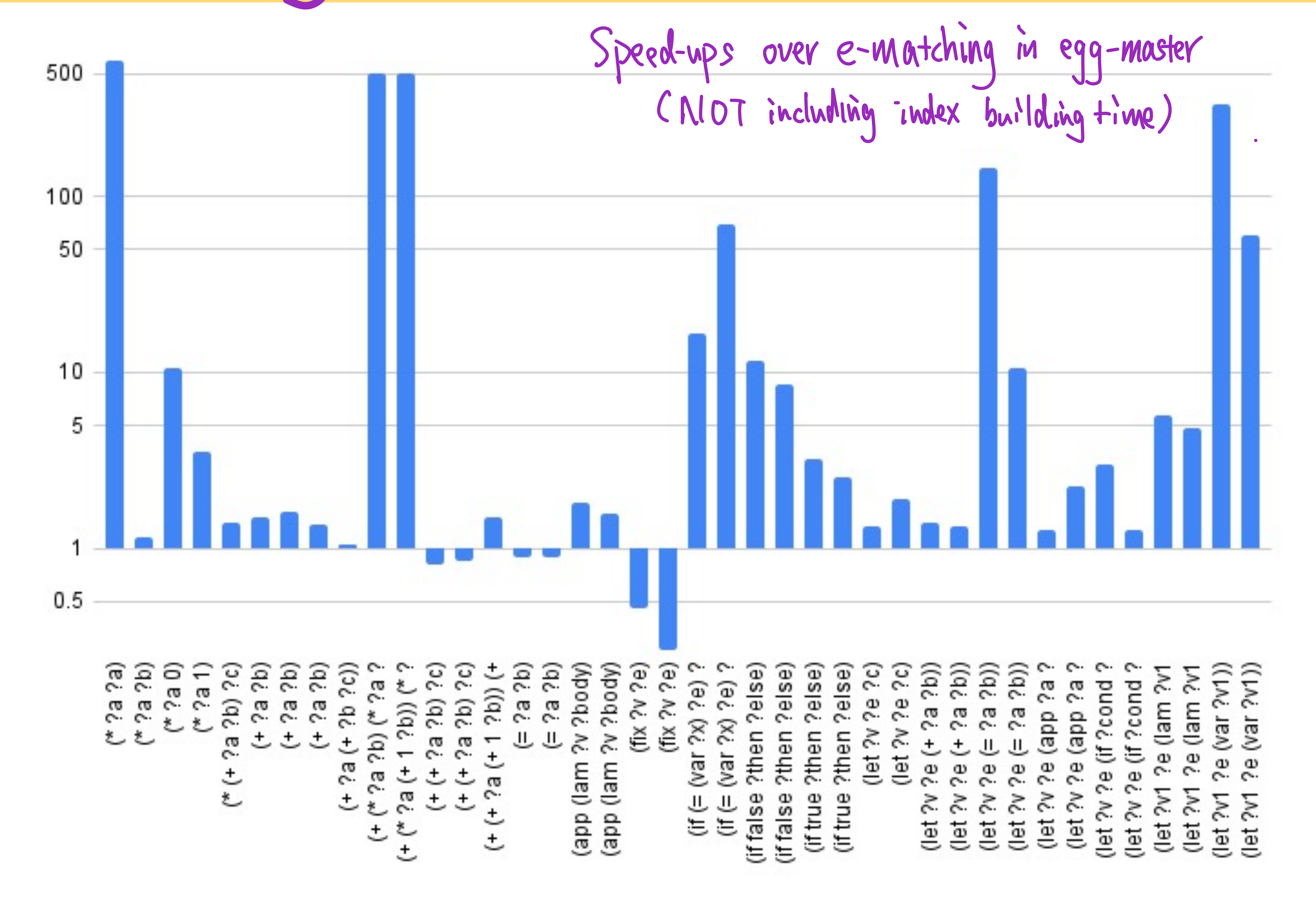
- Collected from egg's math test suite
- Asymptotic speedup on nonlinear patterns
- Caveat: relational e-matching is handwritten in this benchmark

(Preliminary) Evaluations - microbenchmarks

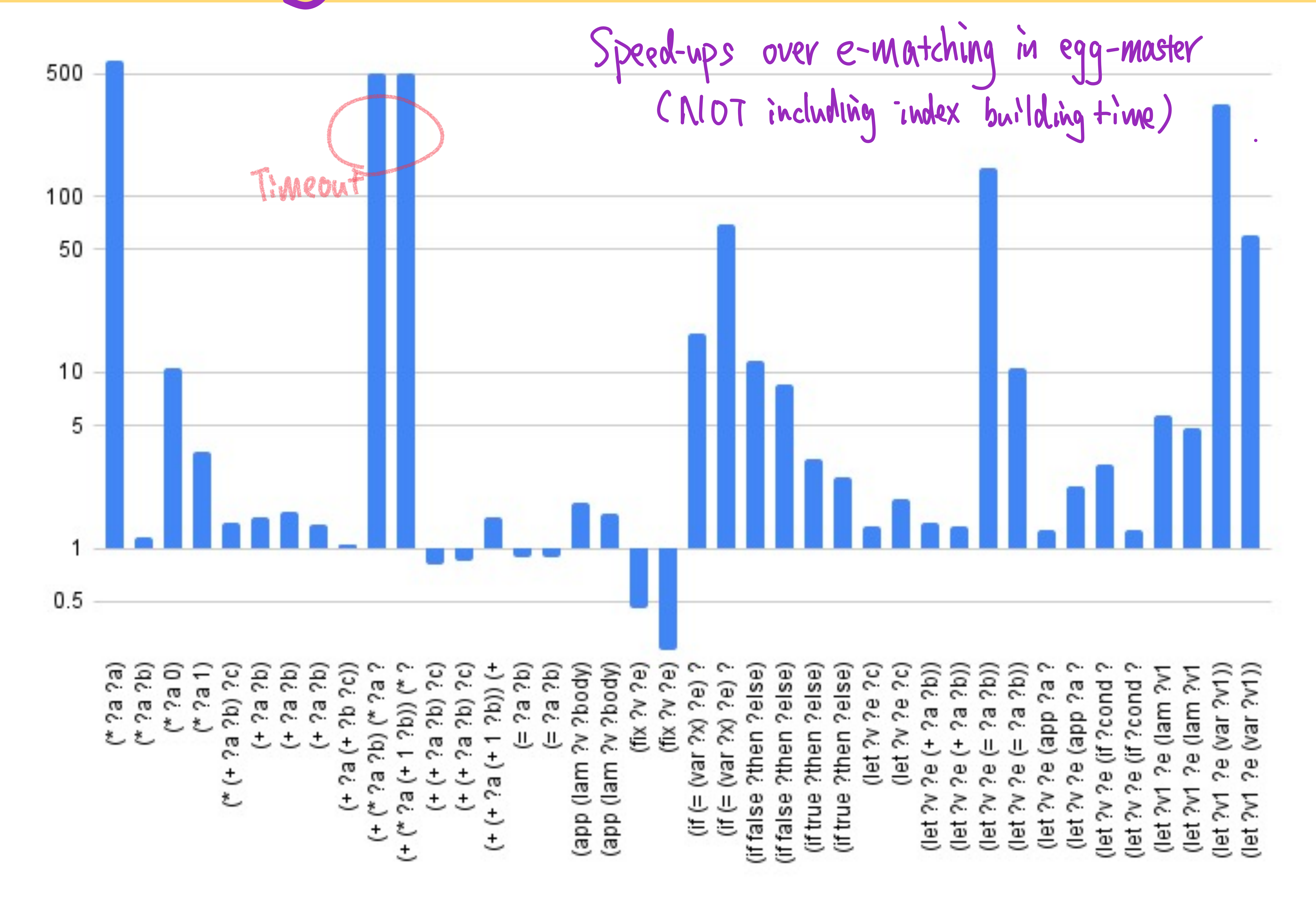
(Preliminary) Evaluations - microbenchmarks



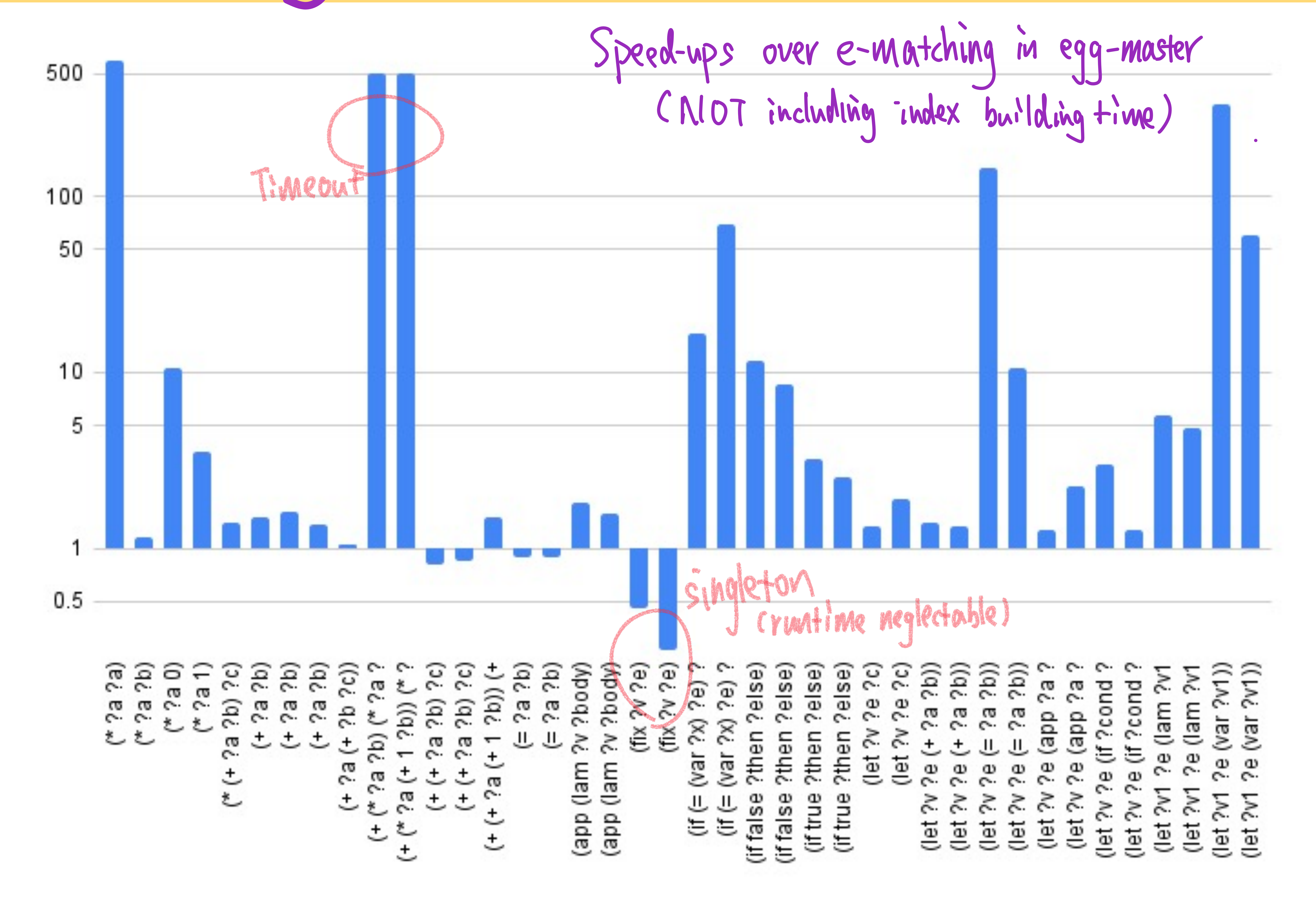
(Preliminary) Evaluations - microbenchmarks



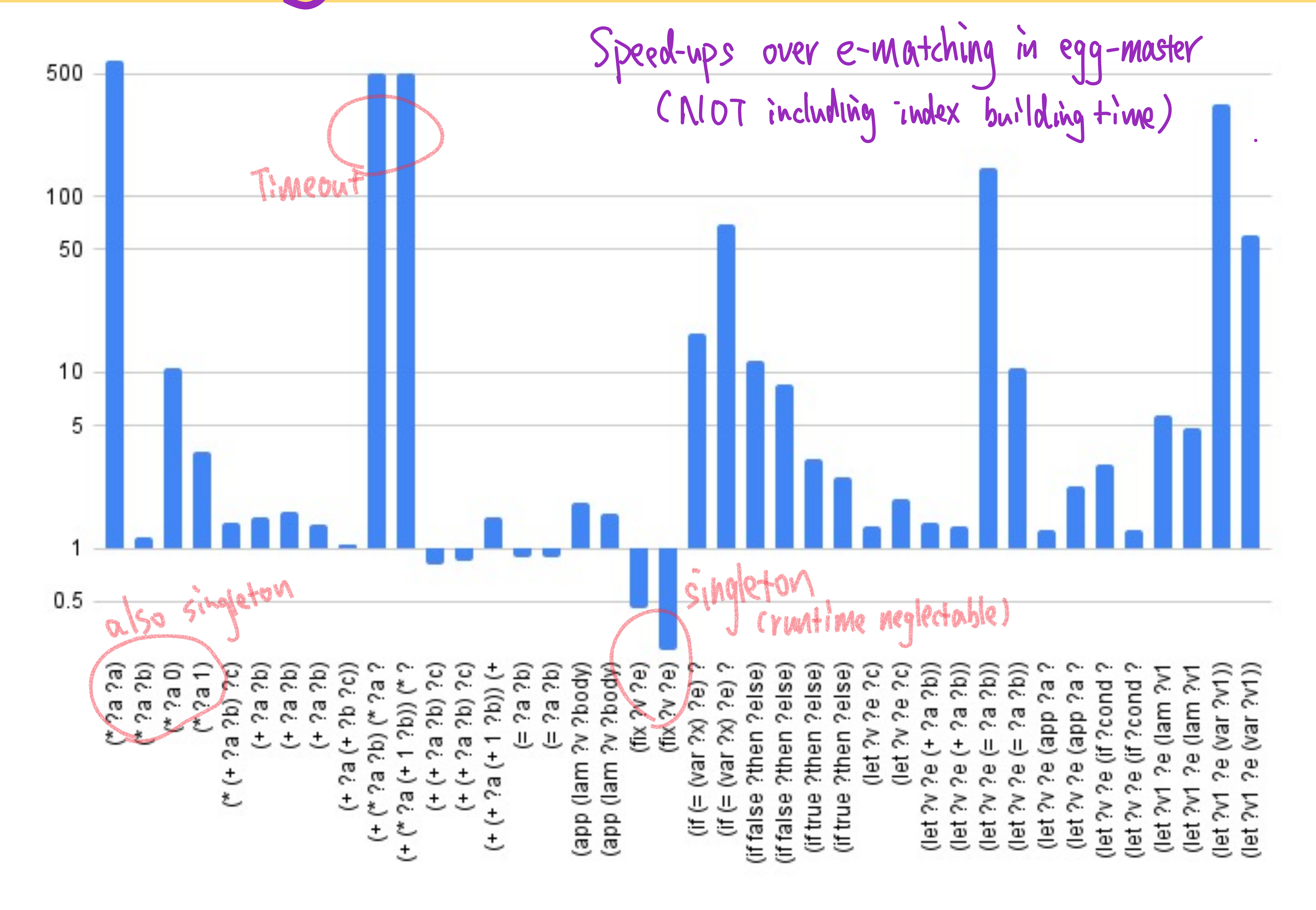
(Preliminary) Evaluations - microbenchmarks



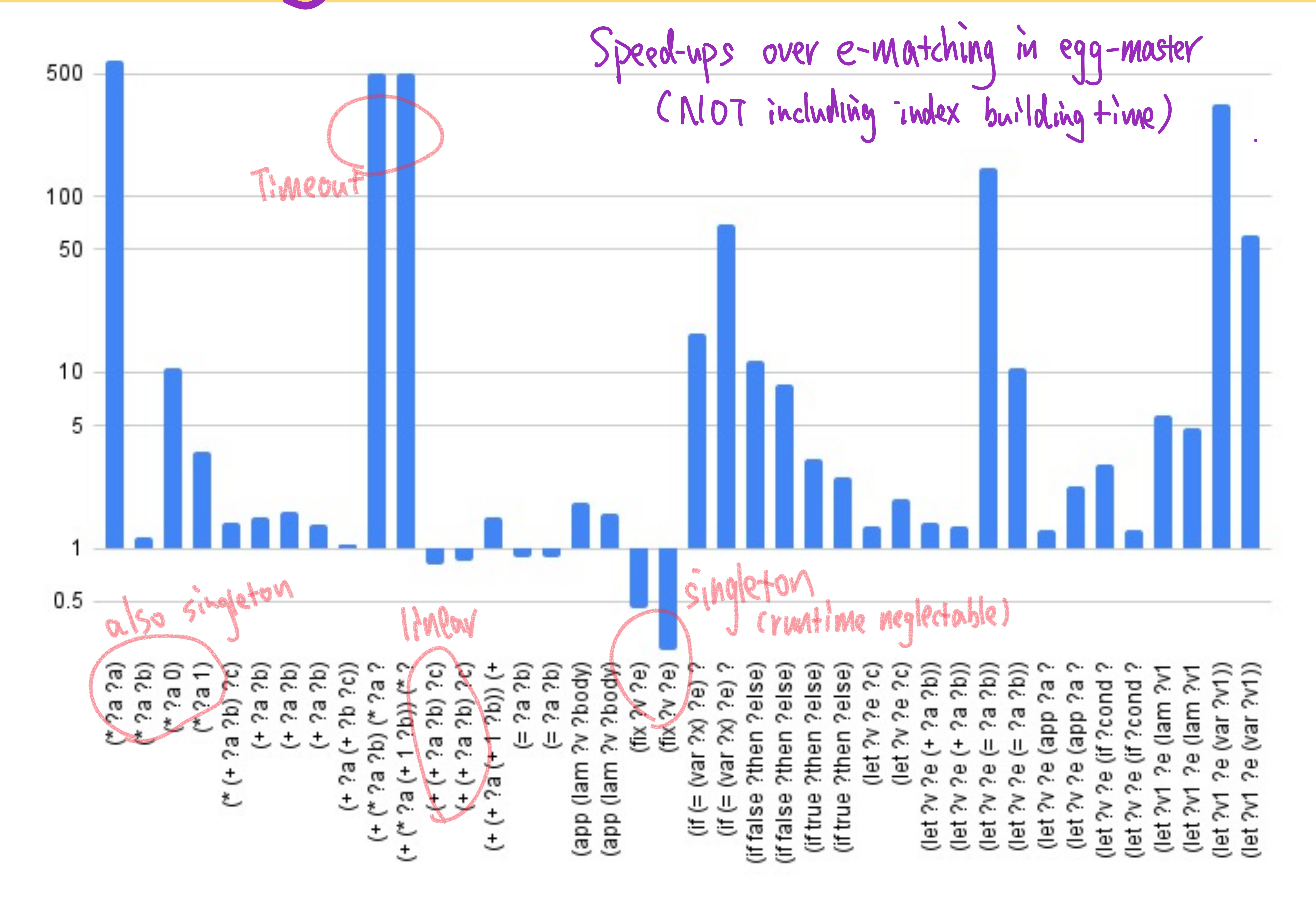
(Preliminary) Evaluations - microbenchmarks



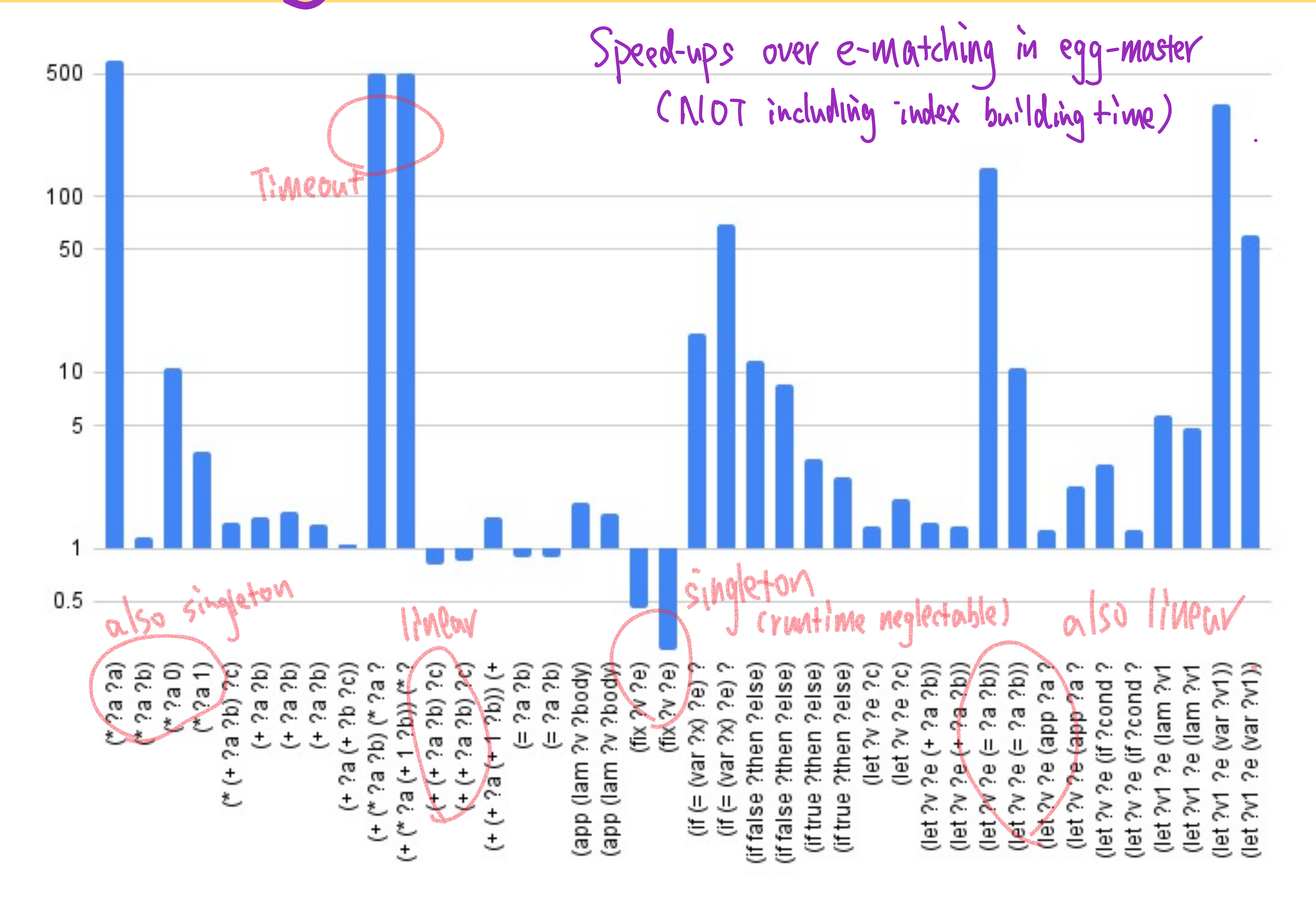
(Preliminary) Evaluations - microbenchmarks



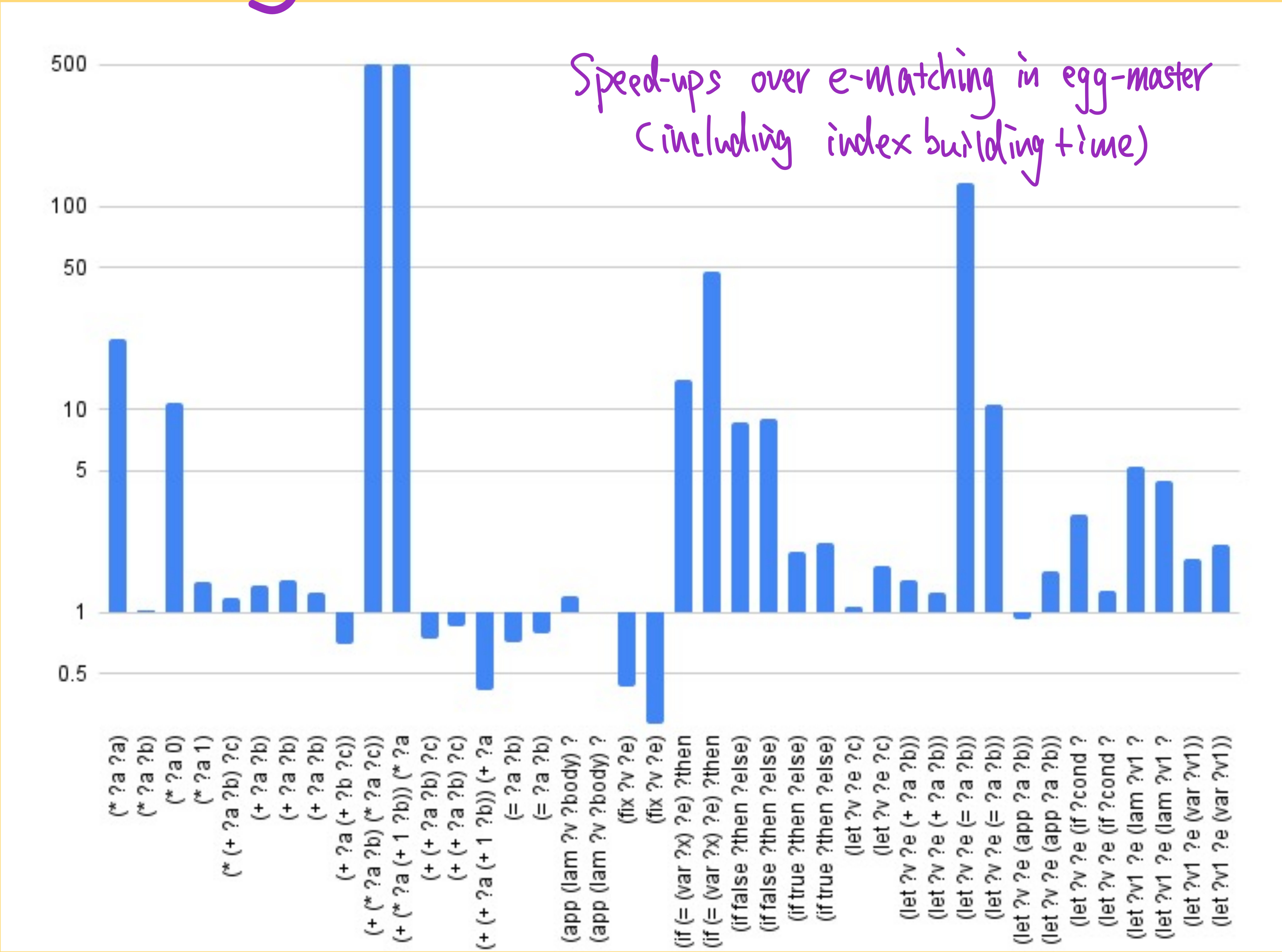
(Preliminary) Evaluations - microbenchmarks



(Preliminary) Evaluations - microbenchmarks



(Preliminary) Evaluations - microbenchmarks



(Preliminary) Evaluations - microbenchmarks

(Preliminary) Evaluations - microbenchmarks

- The more complex, the more efficient

(Preliminary) Evaluations - microbenchmarks

- The more complex, the more efficient
- It's hard to beat backtracking-based e-matching on linear patterns.

(Preliminary) Evaluations - microbenchmarks

- The more complex, the more efficient
- It's hard to beat backtracking-based e-matching on linear patterns.
- ↑ not always true: in some cases the optimizer can exploit cardinality skew in the database and make better plans.

(Preliminary) Evaluations - microbenchmarks

- The more complex, the more efficient
- It's hard to beat backtracking-based e-matching on linear patterns.
- ↑ not always true: in some cases the optimizer can exploit cardinality skew in the database and make better plans.
- Building indices takes time.

How about real-world applications?

How about real-world applications?

:(

How about real-world applications?

:(

working on fine-tune the performance
for linear patterns

Future work

Future work

- Functional dependency
  - a tighter worst-case bound

Future work

- Functional dependency
  - a tighter worst-case bound
- Multi-patterns
  - straightforward to support

Future work

- Functional dependency
    - a tighter worst-case bound
- Multi-patterns
    - straightforward to support
- e-graph engine fully on top of relational database
    - interactive e-matching

Future work

- Functional dependency
  - a tighter worst-case bound
- Multi-patterns
  - straightforward to support
- e-graph engine fully on top of relational database
  - interactive e-matching
- datalog with internalized congruence
  - a richer language for e-graph (e.g. transitive closure)
  - a faster datalog

Takeaways

Takeaways

- Relational e-Matching is simpler, faster, and optimal.

Takeaways

- Relational e-Matching is simpler, faster, and optimal.
- Backtracking-based e-matchings is bad because they don't exploit the equality constraints during query planning

Takeaways

- Relational e-Matching is simpler, faster, and optimal.
- Backtracking-based e-matchings is bad because they don't exploit the equality constraints during query planning
- Relational database is a very powerful abstraction.

Takeaways

- Relational e-Matching is simpler, faster, and optimal.
- Backtracking-based e-matchings is bad because they don't exploit the equality constraints during query planning
- Relational database is a very powerful abstraction.
- Needs to fine-tune the performance for easy patterns.