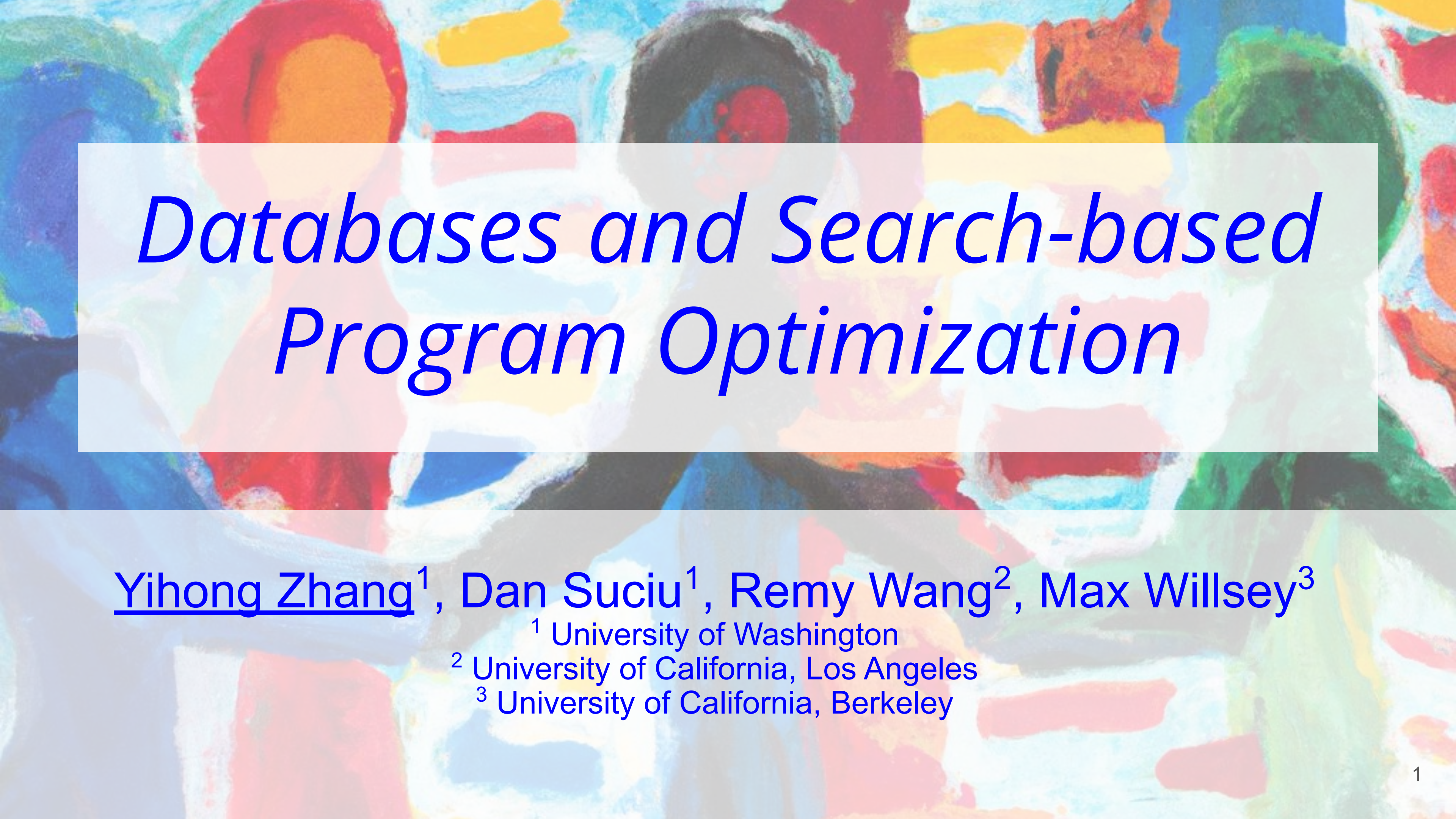


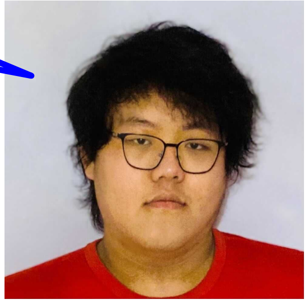
Databases and Search-based Program Optimization

Yihong Zhang[1], Dan Suciu[1], Remy Wang[2], Max Willsey[3]
[1] University of Washington
[2] University of California, Los Angeles
[3] University of California, Berkeley

About me

About me

PhD student at
Univ. of Washington



About me

PhD student at
Univ. of Washington

I work on
<u>Equality Saturation</u>

About me

PhD student at
Univ. of Washington

I work on
<u>Equality Saturation</u>

Program optimization technique used in 50+ projects

- Awards: PLDI ‘15, OOPSLA ‘21, ASPLOS ‘24, POPL ‘24.
- Industry users: Intel, Certora, Bytecode Alliance, ...
- EqSat Papers: VLDB ‘20, SIGMOD ‘22 ‘23, ICDE ‘22, PLDI ‘20 ‘24^{x2}, OOPSLA ‘21 ‘23 ‘24, ASPLOS ‘21 ‘23 ‘24^{x3} ‘25, POPL ‘09 ‘23, ICFP ‘24, CCA ‘21, CCS ‘22, CGO ‘24^{x2} ‘25^{x2}, DAC ‘23^{x2} ‘24, EGRAPHS ‘22 ‘23^{x4}, FCCM ‘22^{x2}, PACT ‘22^{x2} ‘24, DAC ‘23^{x2} 24, FMCAD ‘22, MLSys ‘21, MAPS ‘21, IDDM ‘23, SIGA ‘19, TOG ‘22 ...

About me

PhD student at
Univ. of Washington

I work on
Equality Saturation

Program optimization technique used in 50+ projects

- Awards: PLDI ‘15, OOPSLA ‘21, ASPLOS ‘24, POPL ‘24.
- Industry users: Intel, Certora, Bytecode Alliance, ...
- EqSat Papers: VLDB ‘20, SIGMOD ‘22 ‘23, ICDE ‘22, PLDI ‘20 ‘24^{x2}, OOPSLA ‘21 ‘23 ‘24, ASPLOS ‘21 ‘23 ‘24^{x3} ‘25, POPL ‘09 ‘23, ICFP ‘24, CCA ‘21, CCS ‘22, CGO ‘24^{x2} ‘25^{x2}, DAC ‘23^{x2} ‘24, EGRAPHS ‘22 ‘23^{x4}, FCCM ‘22^{x2}, PACT ‘22^{x2} ‘24, DAC ‘23^{x2} 24, FMCAD ‘22, MLSys ‘21, MAPS ‘21, IDDM ‘23, SIGA ‘19, TOG ‘22 ...

The more I study EqSat,
the more I realize:

It’s just databases!

Optimizing programs using term rewriting

(a * 2) / 2 ⇨ a

Optimizing programs using term rewriting

$(a * 2) / 2$

Optimizing programs using term rewriting

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

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

Optimizing programs using term rewriting

$$(a * 2) / 2 \xrightarrow{(x * y) / z = x * (y / z)} a * (2 / 2) \xrightarrow{x / x = 1} a * 1$$

Optimizing programs using term rewriting

$$(a * 2) / 2 \xrightarrow{(x * y) / z = x * (y / z)} a * (2 / 2) \xrightarrow{x / x = 1} a * 1 \xrightarrow{x * 1 = x} a$$

Optimizing programs using term rewriting

$$(a * 2) / 2 \xrightarrow{(x * y) / z = x * (y / z)} a * (2 / 2) \xrightarrow{x / x = 1} a * 1 \xrightarrow{x * 1 = x} a$$

$$(a * 2) / 2$$

Optimizing programs using term rewriting

$$(a * 2) / 2 \xrightarrow{(x * y) / z = x * (y / z)} a * (2 / 2) \xrightarrow{x / x = 1} a * 1 \xrightarrow{x * 1 = x} a$$

$$(a * 2) / 2 \xrightarrow{x * 2 = x \ll 1} (a \ll 1) / 2$$

Optimizing programs using term rewriting

$$\begin{tikzcd}
(a * 2) / 2 \arrow[r, "(x * y) / z = x * (y / z)"] & a * (2 / 2) \arrow[r, "x / x = 1"] & a * 1 \arrow[r, "x * 1 = x"] & a
\end{tikzcd}$$

$$\begin{tikzcd}
(a * 2) / 2 \arrow[r, "x * 2 = x \ll 1"] & (a \ll 1) / 2 \arrow[r] & ?
\end{tikzcd}$$

Optimizing programs using term rewriting

$$(a * 2) / 2 \xrightarrow{(x * y) / z = x * (y / z)} a * (2 / 2) \xrightarrow{x / x = 1} a * 1 \xrightarrow{x * 1 = x} a$$

$$(a * 2) / 2 \xrightarrow{x * 2 = x \ll 1} (a \ll 1) / 2 \longrightarrow ?$$

$$a \longrightarrow a * 1 \longrightarrow a * 1 * 1 \longrightarrow \ldots$$

Optimizing programs using term rewriting

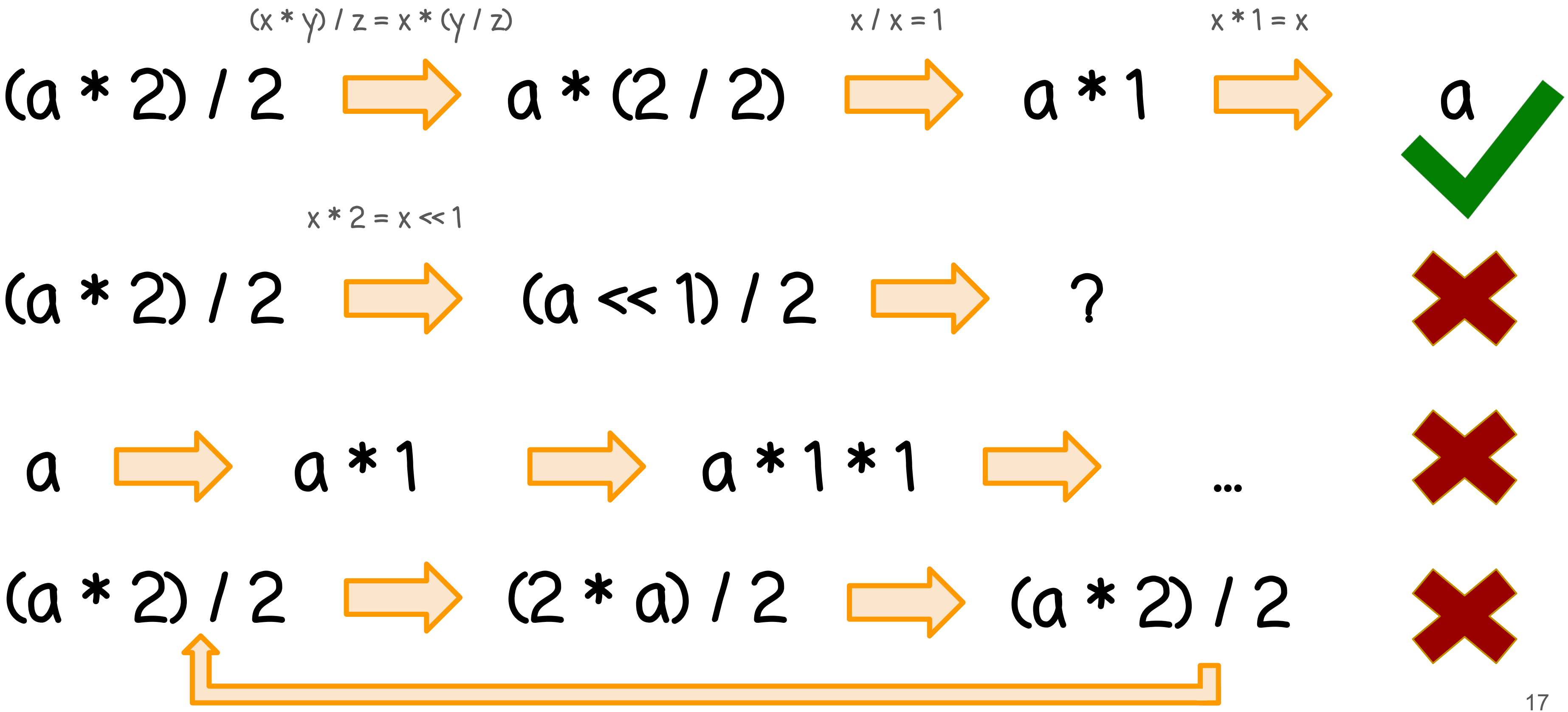
$$(a * 2) / 2 \xrightarrow{(x * y) / z = x * (y / z)} a * (2 / 2) \xrightarrow{x / x = 1} a * 1 \xrightarrow{x * 1 = x} a$$

$$(a * 2) / 2 \xrightarrow{x * 2 = x \ll 1} (a \ll 1) / 2 \longrightarrow ?$$

$$a \longrightarrow a * 1 \longrightarrow a * 1 * 1 \longrightarrow \ldots$$

$$\begin{tikzcd}
(a * 2) / 2 \arrow[r] & (2 * a) / 2 \arrow[r] & (a * 2) / 2 \arrow[ll, bend left]
\end{tikzcd}$$

Optimizing programs using term rewriting



$(a * 2) / 2 \xrightarrow{(x * y) / z = x * (y / z)} a * (2 / 2) \xrightarrow{x / x = 1} a * 1 \xrightarrow{x * 1 = x} a$ ✓

$(a * 2) / 2 \xrightarrow{x * 2 = x << 1} (a << 1) / 2 \rightarrow ?$ ✗

$a \rightarrow a * 1 \rightarrow a * 1 * 1 \rightarrow \ldots$ ✗

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

Optimizing programs using term rewriting

$$(a * 2) / 2 \xrightarrow{(x * y) / z = x * (y / z)} a * (2 / 2) \xrightarrow{x / x = 1} a * 1 \xrightarrow{x * 1 = x} a$$ ✓

$$(a * 2) / 2 \xrightarrow{x * 2 = x \ll 1} (a \ll 1) / 2 \xrightarrow{} ?$$ ✗

$$a \xrightarrow{} a * 1 \xrightarrow{} a * 1 * 1 \xrightarrow{} \ldots$$ ✗

$$\begin{tikzcd}(a * 2) / 2 \arrow[r] & (2 * a) / 2 \arrow[r] & (a * 2) / 2 \arrow[ll, bend left]\end{tikzcd}$$ 🤔

Optimizing programs using term rewriting

$$(a * 2) / 2 \xrightarrow{(x * y) / z = x * (y / z)} a * (2 / 2) \xrightarrow{x / x = 1} a * 1 \xrightarrow{x * 1 = x} a$$ ✓

x * 2 = x << 1

$(a *$

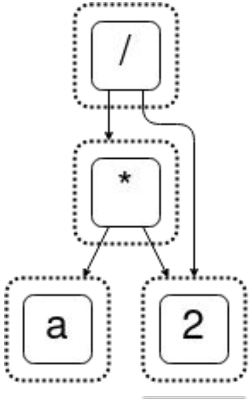
Equality Saturation: apply all the rules all the time!

✗

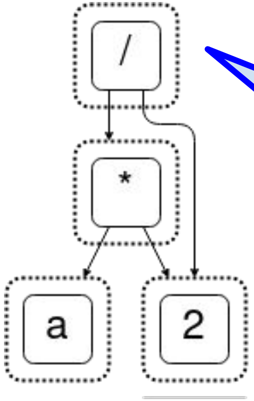
$$a \longrightarrow a * 1 \longrightarrow a * 1 * 1 \longrightarrow \ldots$$ ✗

$$\begin{tikzcd}(a * 2) / 2 \arrow[r] & (2 * a) / 2 \arrow[r] & (a * 2) / 2 \arrow[ll, bend left]\end{tikzcd}$$ 🤔 ✗

E-graphs and Equality saturation

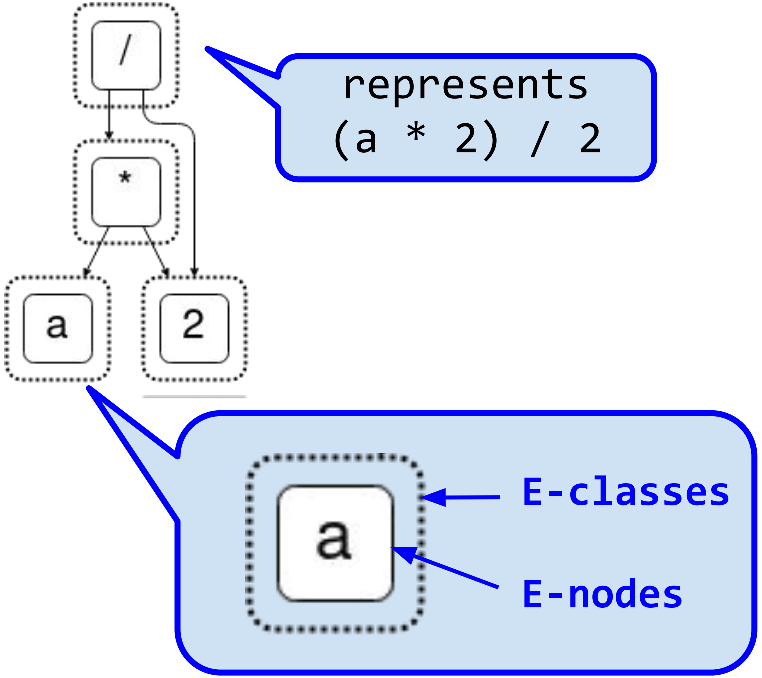


E-graphs and Equality saturation



represents
`(a * 2) / 2`

E-graphs and Equality saturation

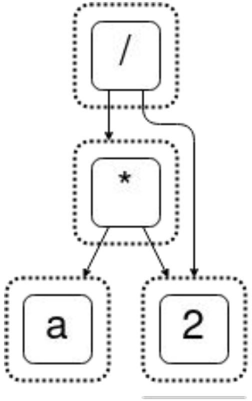


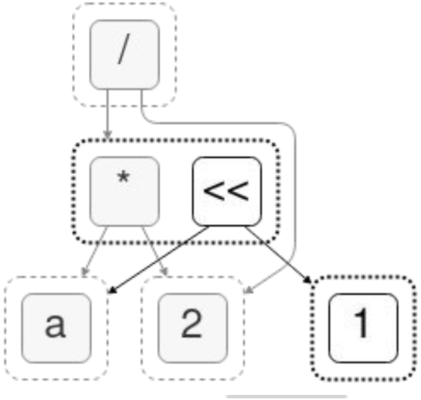
represents
(a * 2) / 2

E-classes

E-nodes

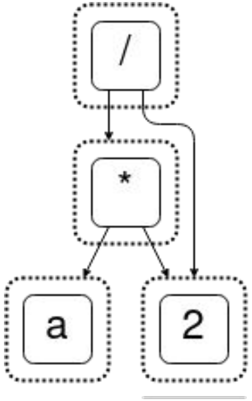
E-graphs and Equality saturation

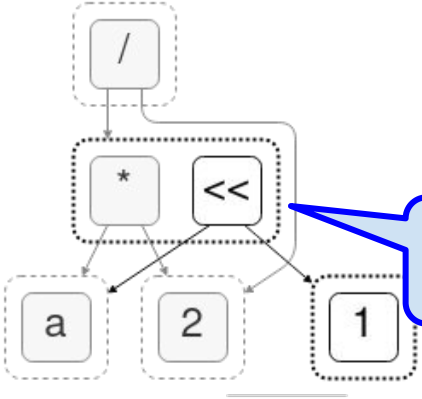




```
x * 2 => x << 1
```

E-graphs and Equality saturation

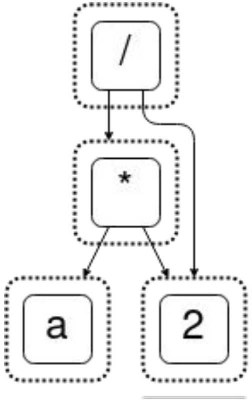


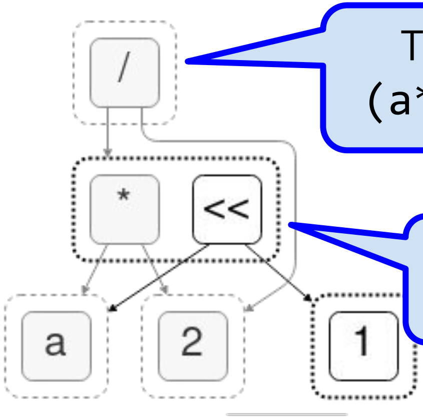


This e-class *represents*
`a * 2` **and** `a << 1`

```
x * 2 => x << 1
```

E-graphs and Equality saturation



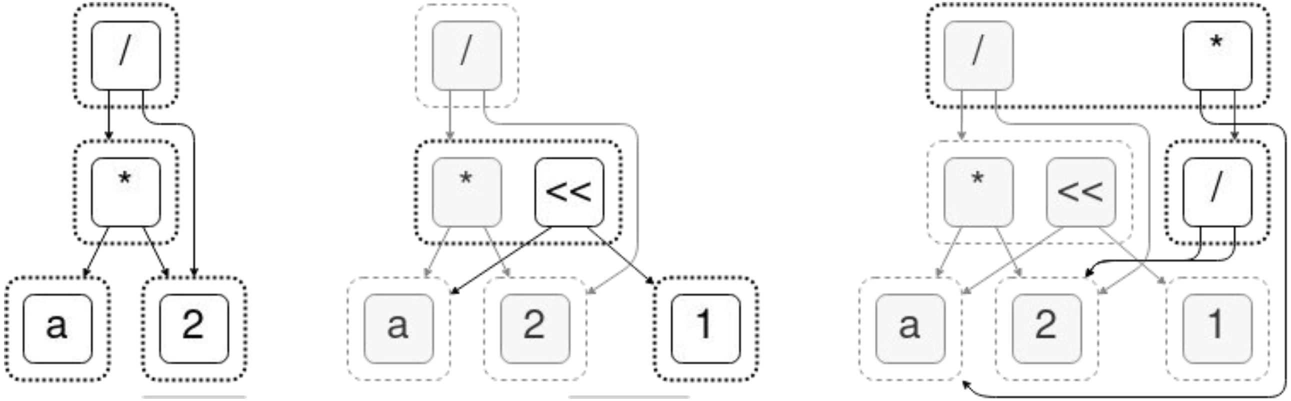


This e-class *represents* `(a*2)/2` **and** `(a<<1)/2`

This e-class *represents* `a * 2` **and** `a << 1`

```
x * 2 => x << 1
```

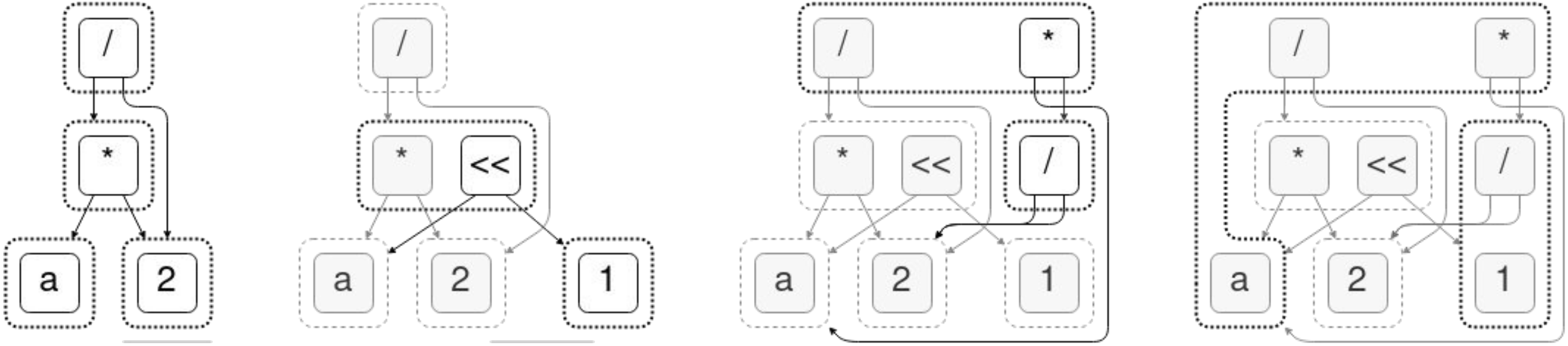
E-graphs and Equality saturation



`x * 2 => x << 1`

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

E-graphs and Equality saturation



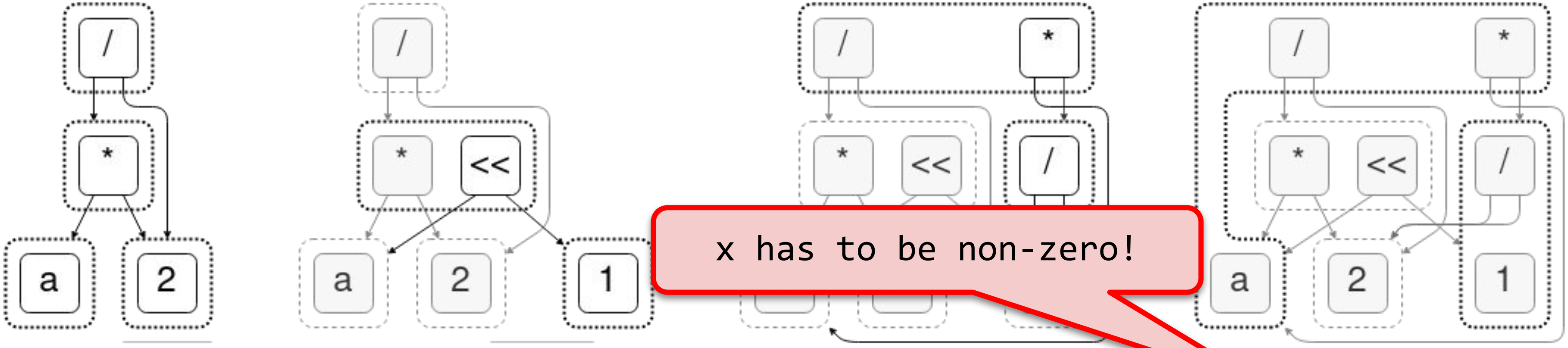
```
x * 2 => x << 1
```

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

```
x / x => 1
x * 1 => x
```

```
loop until
fixpoint / timeout!
```

E-graphs and Equality saturation



x has to be non-zero!

x * 2 => x << 1

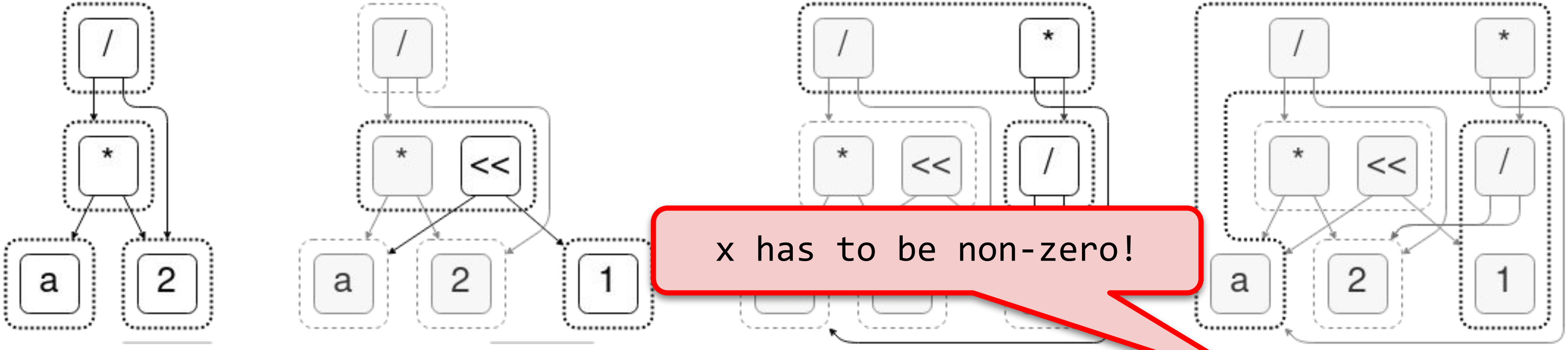
(x * y) / z => x * (y / z)

x / x => 1

x * 1 => x

loop until
fixpoint / timeout!

E-graphs and Equality saturation



x * 2 => x << 1

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

x / x => 1

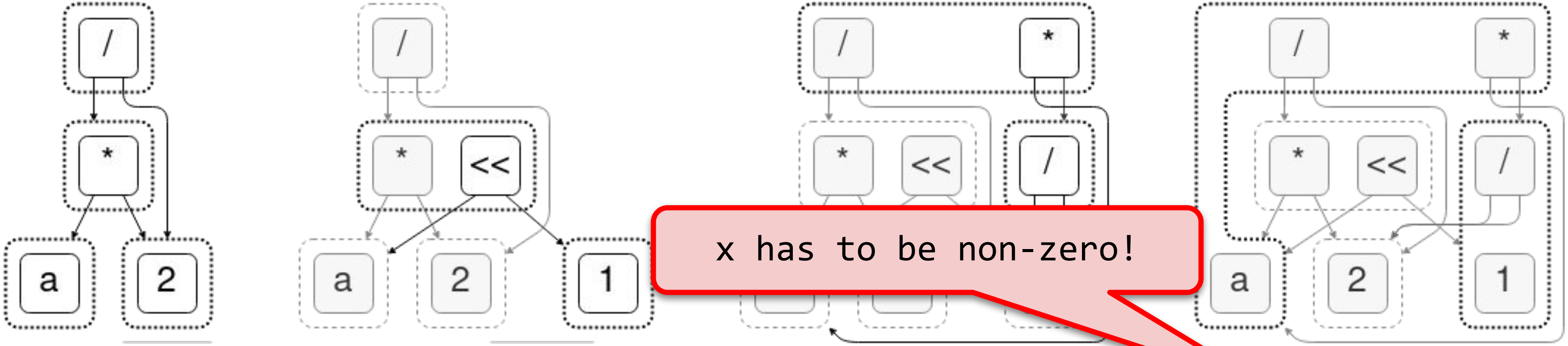
x * 1 => x

x has to be non-zero!

0 ∉ rangeOf(2)

loop until
fixpoint / timeout!

E-graphs and Equality saturation



x has to be non-zero!

x * 2 => x << 1

“E-class analyses”

* (y / z)

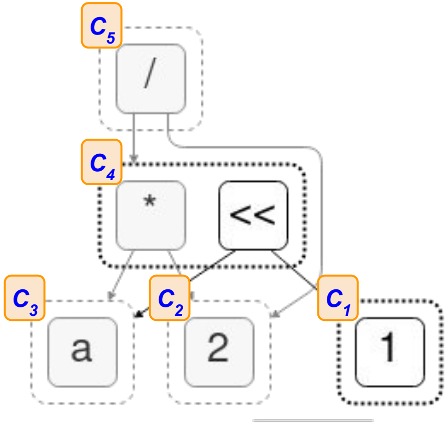
x / x => 1

x * 1 => x

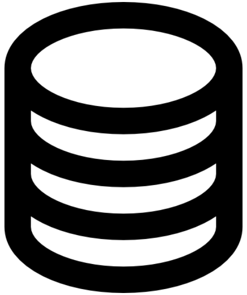
0 ∉ rangeOf(2)

loop until
fixpoint / timeout!

E-graphs and Equality saturation

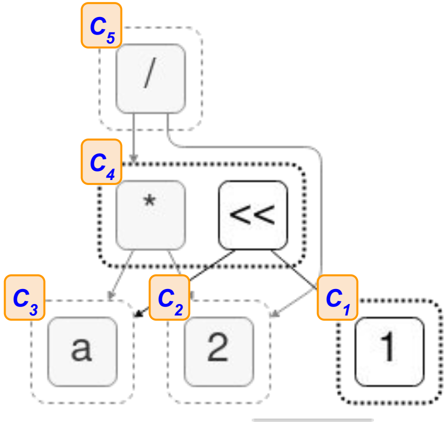


=



```
x * 2 => x << 1
```

E-graphs and Equality saturation



```
x * 2 => x << 1
```

=

Table: number

child	out
1	C_1
2	C_2

Table: variable

child	out
“a”	C_3

Table: *

ch$_1$	ch$_2$	out
C_3	C_2	C_4

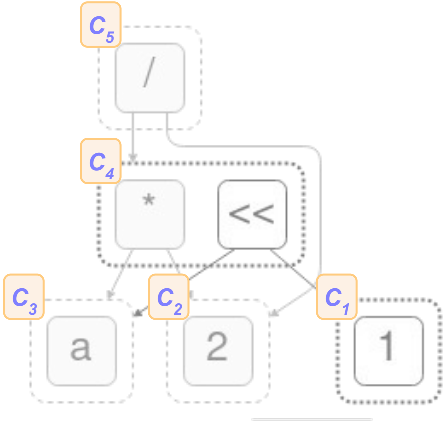
Table: <<

ch$_1$	ch$_2$	out
C_3	C_1	C_4

Table: /

ch$_1$	ch$_2$	out
C_4	C_2	C_5

E-graphs and Equality saturation



`x * 2 => x << 1`

=

Table: number

child	out
1	c_1
2	c_2

Table: variable

child	out
“a”	c_3

Table: *

ch_1	ch_2	out
c_3	c_2	c_4

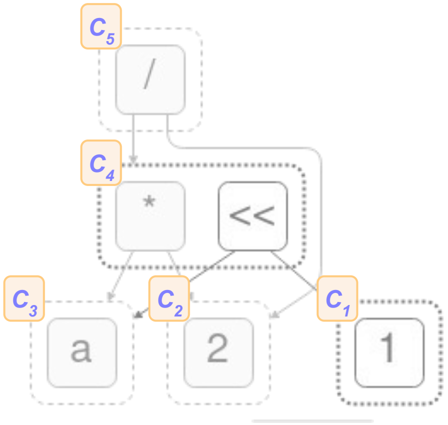
Table: <<

ch_1	ch_2	out
c_3	c_1	c_4

Table: /

ch_1	ch_2	out
c_4	c_2	c_5

E-graphs and Equality saturation



=

Table: number

child	out
1	C_1
2	C_2

Table: variable

child	out
“a”	C_3

Table: *

ch_1	ch_2	out
C_3	C_2	C_4

Table: <<

ch_1	ch_2	out
C_3	C_1	C_4

Table: /

ch_1	ch_2	out
C_4	C_2	C_5

```
x * 2 => x << 1
```

$<<(x, C_1, r)\ :\!-\ *(x, C_2, r)$

E-graphs and Equality saturation

E-graphs		Databases
E-nodes		Tuples
E-classes		Labeled null
Rewrite rules	🤝	Tuple-generating dep. (TGD)
Congruence closure		Functional dep. (FD)
E-class analyses E-graph extraction		Monotonic Aggregation
Equality Saturation		**Chase**

E-graphs and Equality saturation

E-graphs		Databases
E-nodes		Tuples
E-classes		Labeled null
Rewrite rules	🤝	Tuple-generating dep. (TGD)
Congruence closure		Functional dep. (FD)
E-class analyses E-graph extraction		Monotonic Aggregation
Equality Saturation		**Chase**

More details in our ICDT paper (Suciu, Wang, Zhang)

It’s also true the other way around!

Volcano/Cascades is also EqSat!

Volcano/Cascades is also EqSat!

Volcano and Cascades are query optimization frameworks that combines rule-based optimization and cost-based optimization.

Volcano/Cascades is also EqSat!

Volcano and Cascades are query optimization frameworks that combines rule-based optimization and cost-based optimization.

Key ideas

- apply rewrite rules over a memo table data structure
- use a cost model to select the best query plan

Volcano/Cascades is also EqSat!

Volcano and Cascades are query optimization frameworks that combines rule-based optimization and cost-based optimization.

Key ideas

E-graph

- apply rewrite rules over a memo table data structure
- use a cost model to select the best query plan

Volcano/Cascades is also EqSat!

Volcano and Cascades are query optimization frameworks that combines rule-based optimization and cost-based optimization.

Key ideas

E-graph

- apply rewrite rules over a memo table data structure
- use a cost model to select the best query plan

E-graph Extraction

Volcano/Cascades is also EqSat!

Volcano and Cascades are query optimization frameworks that combines rule-based optimization and cost-based optimization.

Key ideas

E-graph

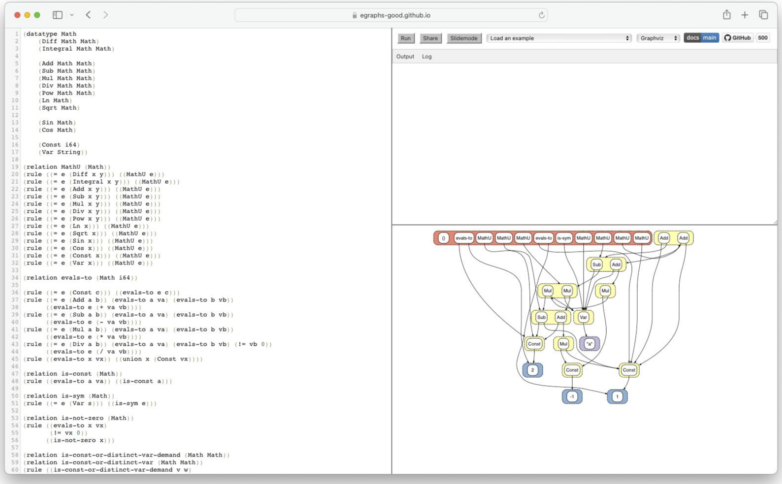
- apply rewrite rules over a memo table data structure
- use a cost model to select the best query plan

E-graph Extraction

🔥Hot take: EqSat is a more principled framework

Case study:
Rule- and cost-based optimization of matrix expressions

https://github.com/egraphs-good/egglog/tree/icdt-db-x-demo



```
(datatype Math
    (Diff Math Math)
    (Integral Math Math)

    (Add Math Math)
    (Sub Math Math)
    (Mul Math Math)
    (Div Math Math)
    (Pow Math Math)
    (Ln Math)
    (Sqrt Math)

    (Sin Math)
    (Cos Math)

    (Const i64)
    (Var String))

(relation MathU (Math))
(rule ((= e (Diff x y))) ((MathU e)))
(rule ((= e (Integral x y))) ((MathU e)))
(rule ((= e (Add x y))) ((MathU e)))
(rule ((= e (Sub x y))) ((MathU e)))
(rule ((= e (Mul x y))) ((MathU e)))
(rule ((= e (Div x y))) ((MathU e)))
(rule ((= e (Pow x y))) ((MathU e)))
(rule ((= e (Ln x))) ((MathU e)))
(rule ((= e (Sqrt x))) ((MathU e)))
(rule ((= e (Sin x))) ((MathU e)))
(rule ((= e (Cos x))) ((MathU e)))
(rule ((= e (Const x))) ((MathU e)))
(rule ((= e (Var x))) ((MathU e)))

(relation evals-to (Math i64))

(rule ((= e (Const c))) ((evals-to e c)))
(rule ((= e (Add a b)) (evals-to a va) (evals-to b vb))
      ((evals-to e (+ va vb))))
(rule ((= e (Sub a b)) (evals-to a va) (evals-to b vb))
      ((evals-to e (- va vb))))
(rule ((= e (Mul a b)) (evals-to a va) (evals-to b vb))
      ((evals-to e (* va vb))))
(rule ((= e (Div a b)) (evals-to a va) (evals-to b vb) (!= vb 0))
      ((evals-to e (/ va vb))))
(rule ((evals-to x vx)) ((union x (Const vx))))

(relation is-const (Math))
(rule ((evals-to a va)) ((is-const a)))

(relation is-sym (Math))
(rule ((= e (Var s))) ((is-sym e)))

(relation is-not-zero (Math))
(rule ((evals-to x vx)
       (!= vx 0))
      ((is-not-zero x)))

(relation is-const-or-distinct-var-demand (Math Math))
(relation is-const-or-distinct-var (Math Math))
(rule ((is-const-or-distinct-var-demand v w)
(sort Mat)
(function Matrix (String String String) Mat)
(function Prod (Mat Mat) Mat)
(function Agg (String Mat) Mat)
```

```
(sort Mat)
(function Matrix (String String String) Mat)
(function Prod (Mat Mat) Mat)
(function Agg (String Mat) Mat)
```

Table: Matrix

ch_1	ch_2	ch_3	out

Table: Prod

ch_1	ch_2	out

Table: Agg

ch_1	ch_2	out

```
(sort Mat)
(function Matrix (String String String) Mat)
(function Prod (Mat Mat) Mat)
(function Agg (String Mat) Mat)

(let A (Matrix "A" "i" "j")) ;; A_{i,j}
(let B (Matrix "B" "j" "k")) ;; B_{j,k}
(let C (Matrix "C" "k" "l")) ;; C_{k,l}
;; Σ_k Σ_j (AB)C
(let ABC (Agg "k" (Agg "j"
             (Prod (Prod A B) C))))
```

Table: Matrix

ch_1	ch_2	ch_3	out

Table: Prod

ch_1	ch_2	out

Table: Agg

ch_1	ch_2	out

```
(sort Mat)
(function Matrix (String String String) Mat)
(function Prod (Mat Mat) Mat)
(function Agg (String Mat) Mat)

(let A (Matrix "A" "i" "j")) ;; A_{i,j}
(let B (Matrix "B" "j" "k")) ;; B_{j,k}
(let C (Matrix "C" "k" "l")) ;; C_{k,l}
;; Σ_k Σ_j (AB)C
(let ABC (Agg "k" (Agg "j"
            (Prod (Prod A B) C))))
```

Table: Matrix

ch_1	ch_2	ch_3	out
“A”	“i”	“j”	c_A
“B”	“j”	“k”	c_B
“C”	“k”	“l”	c_C

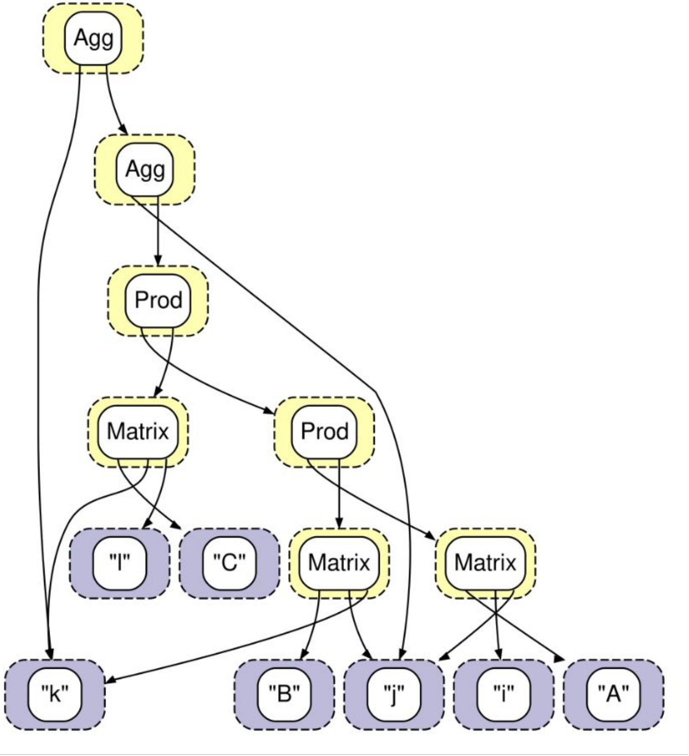
Table: Prod

ch_1	ch_2	out
c_A	c_B	c_{AB}
c_{AB}	c_C	c_{ABC}

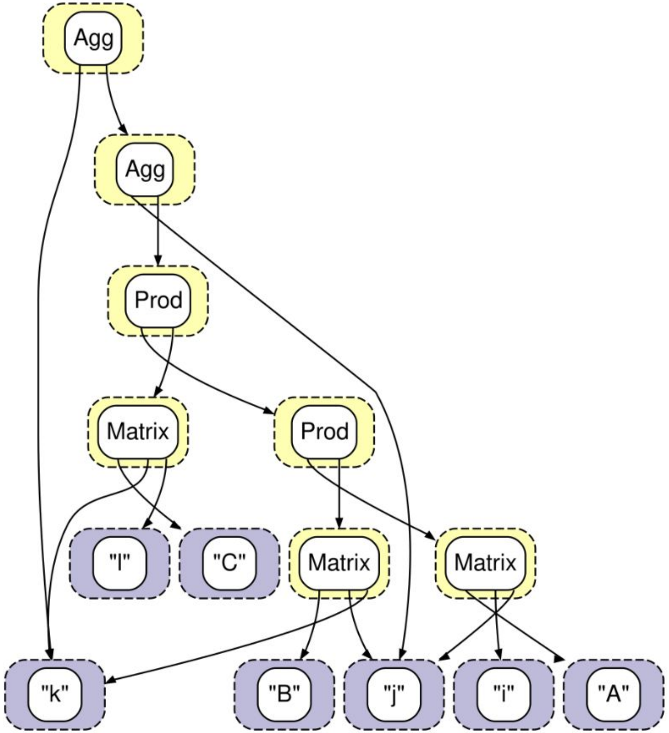
Table: Agg

ch_1	ch_2	out
“j”	c_{ABC}	c_j
“k”	c_j	c_k

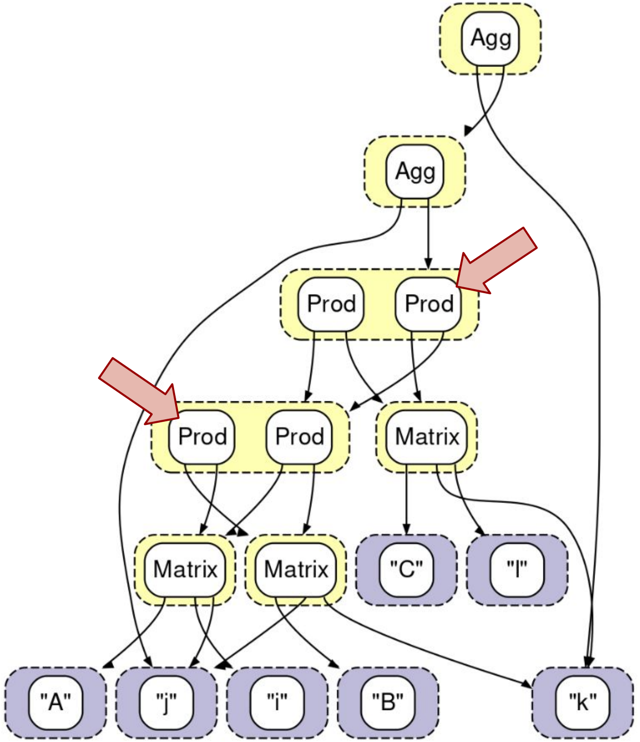
```
(sort Mat)
(function Matrix (String String String) Mat)
(function Prod (Mat Mat) Mat)
(function Agg (String Mat) Mat)

(let A (Matrix "A" "i" "j")) ;; A_{i,j}
(let B (Matrix "B" "j" "k")) ;; B_{j,k}
(let C (Matrix "C" "k" "l")) ;; C_{k,l}
;; Σ_k Σ_j (AB)C
(let ABC (Agg "k" (Agg "j"
            (Prod (Prod A B) C))))
```



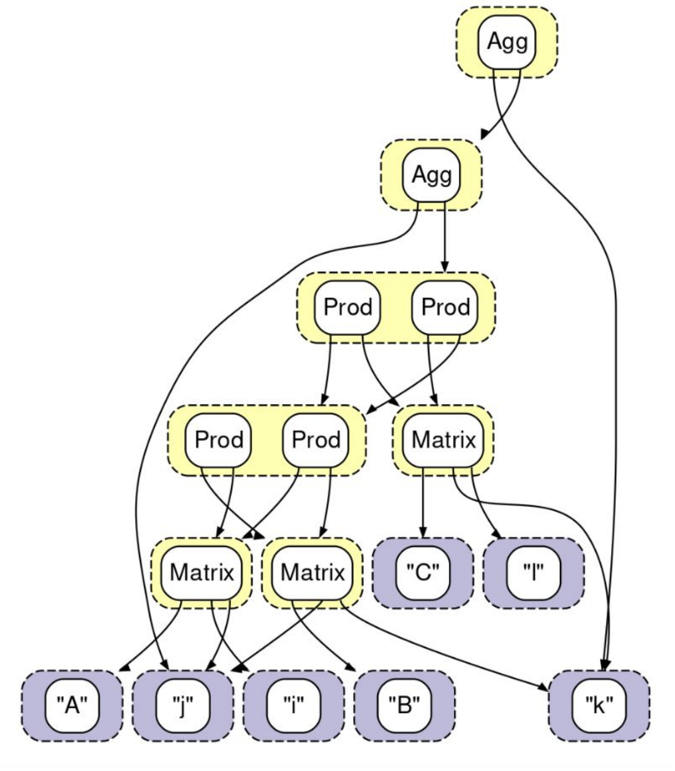
```
;; commutativity
(rewrite (Prod x y) (Prod y x))
```



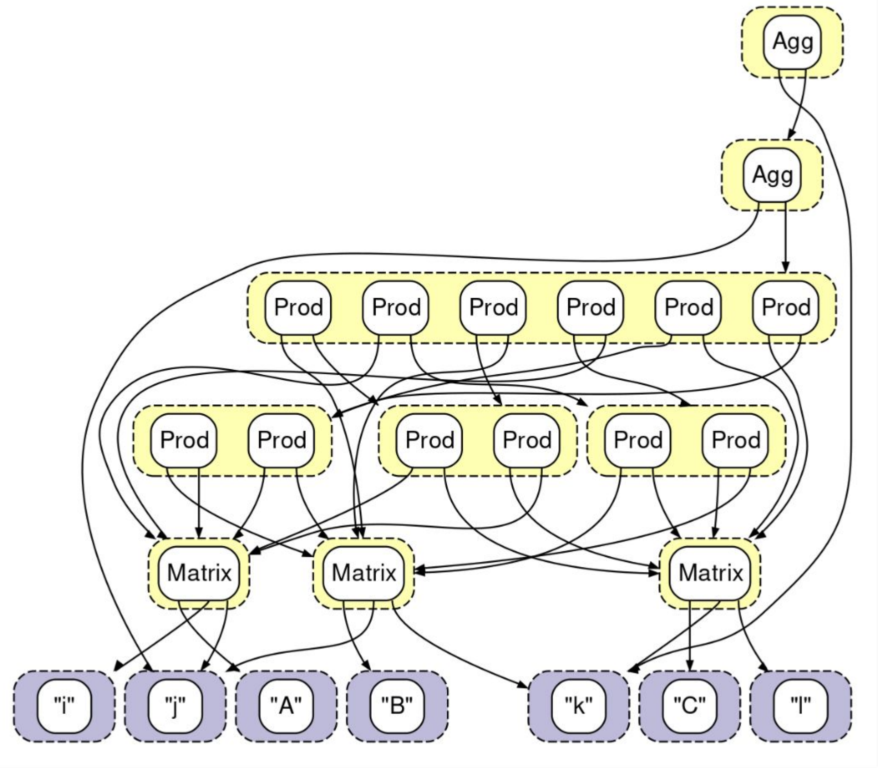
```
;; commutativity
(rewrite (Prod x y) (Prod y x))
```



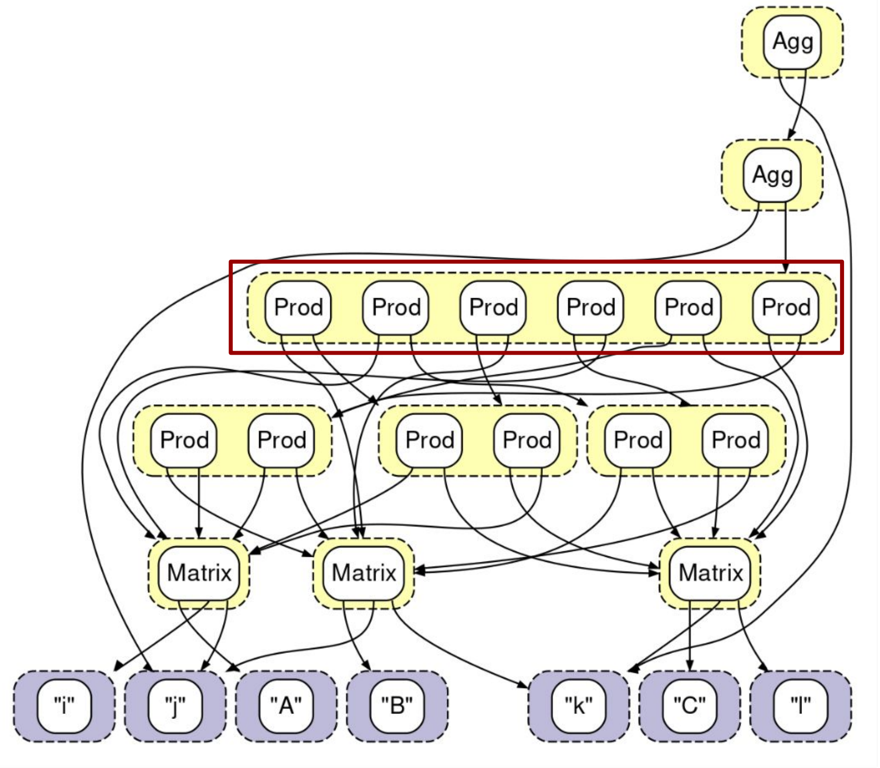
```
;; commutativity
(rewrite (Prod x y) (Prod y x))
;; associativity
(rewrite (Prod (Prod x y) z)
         (Prod x (Prod y z)))
```



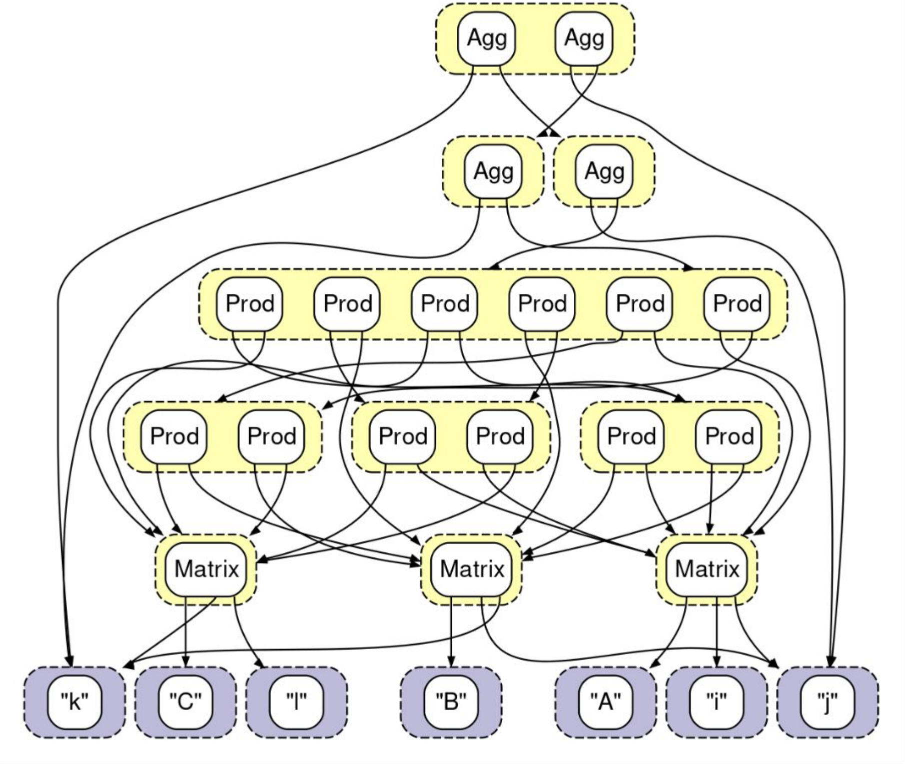
```
;; commutativity
(rewrite (Prod x y) (Prod y x))
;; associativity
(rewrite (Prod (Prod x y) z)
         (Prod x (Prod y z)))
```



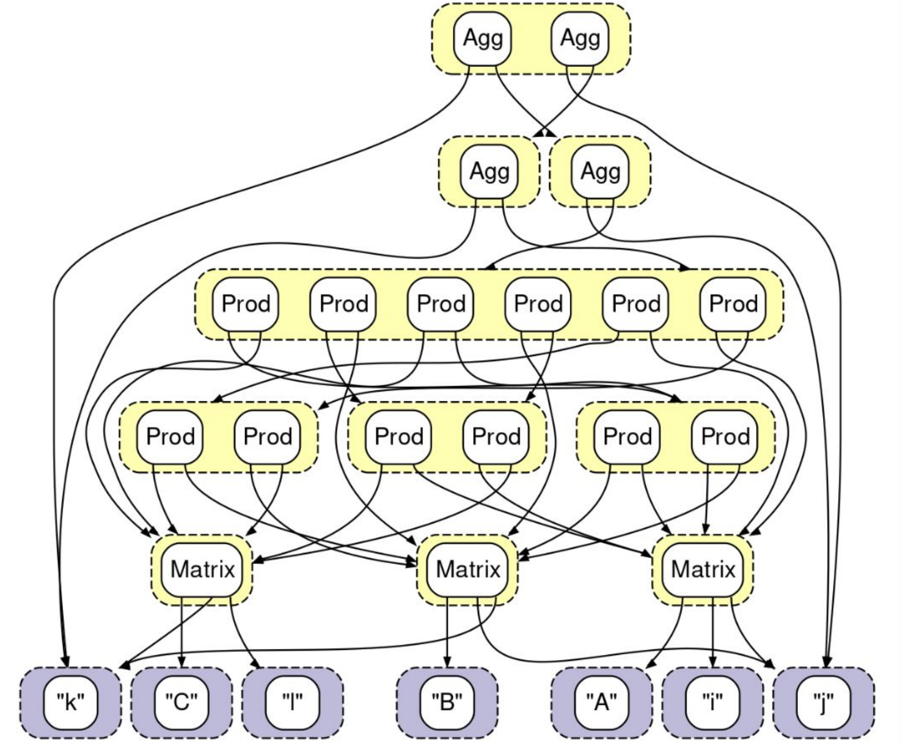
```
;; commutativity
(rewrite (Prod x y) (Prod y x))
;; associativity
(rewrite (Prod (Prod x y) z)
         (Prod x (Prod y z)))
```



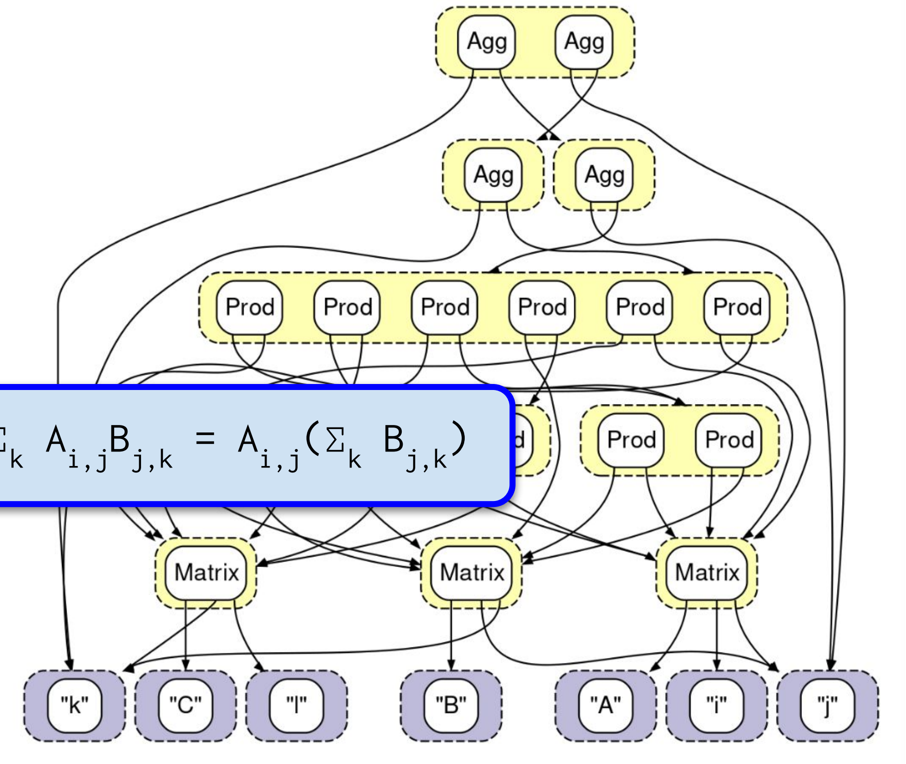
```
;; commutativity
(rewrite (Prod x y) (Prod y x))
;; associativity
(rewrite (Prod (Prod x y) z)
         (Prod x (Prod y z)))
;; commuting aggregation
(rewrite (Agg v1 (Agg v2 e))
         (Agg v2 (Agg v1 e)))
```



```
;; commutativity
(rewrite (Prod x y) (Prod y x))
;; associativity
(rewrite (Prod (Prod x y) z)
         (Prod x (Prod y z)))
;; commuting aggregation
(rewrite (Agg v1 (Agg v2 e))
         (Agg v2 (Agg v1 e)))
;; pushing down aggregation
(rewrite (Agg v (Prod x y))
         (Prod (Agg v x) y)
  :when ((∉ v (vars-of y))))
(rewrite (Agg v (Prod x y))
         (Prod x (Agg v y))
  :when ((∉ v (vars-of x)))
```

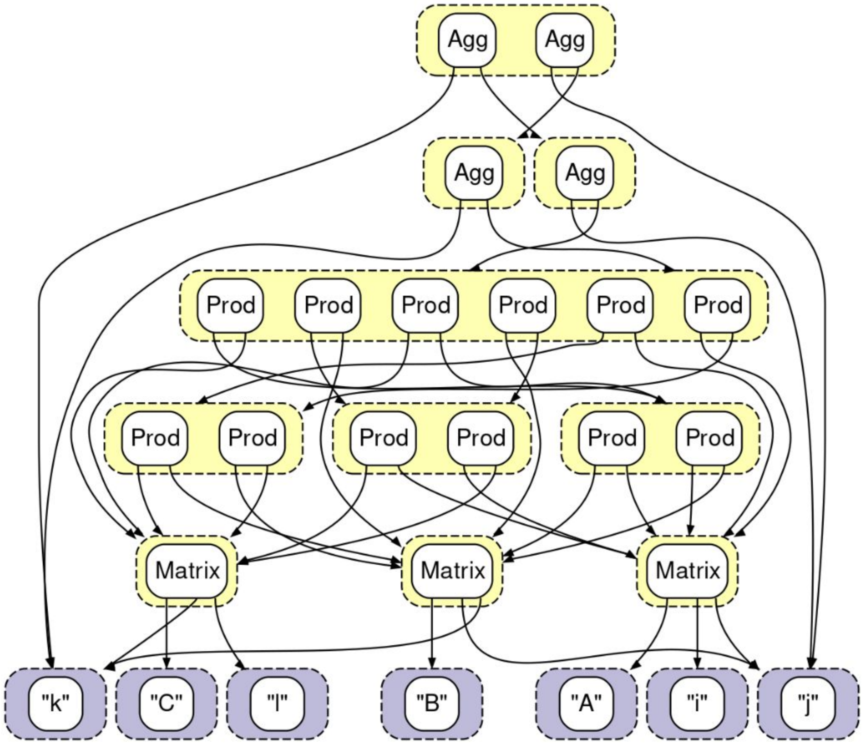


```
;; commutativity
(rewrite (Prod x y) (Prod y x))
;; associativity
(rewrite (Prod (Prod x y) z)
         (Prod x (Prod y z)))
;; commuting aggregation
(rewrite (Agg v1 (Agg v2 e))
         (Agg v2 (Agg v1 e)))
;; pushing down aggregation
(rewrite (Agg v (Prod x y))
         (Prod (Agg v x) y)
  :when ((∉ v (vars-of y))))
(rewrite (Agg v (Prod x y))
         (Prod x (Agg v y))
  :when ((∉ v (vars-of x)))
```

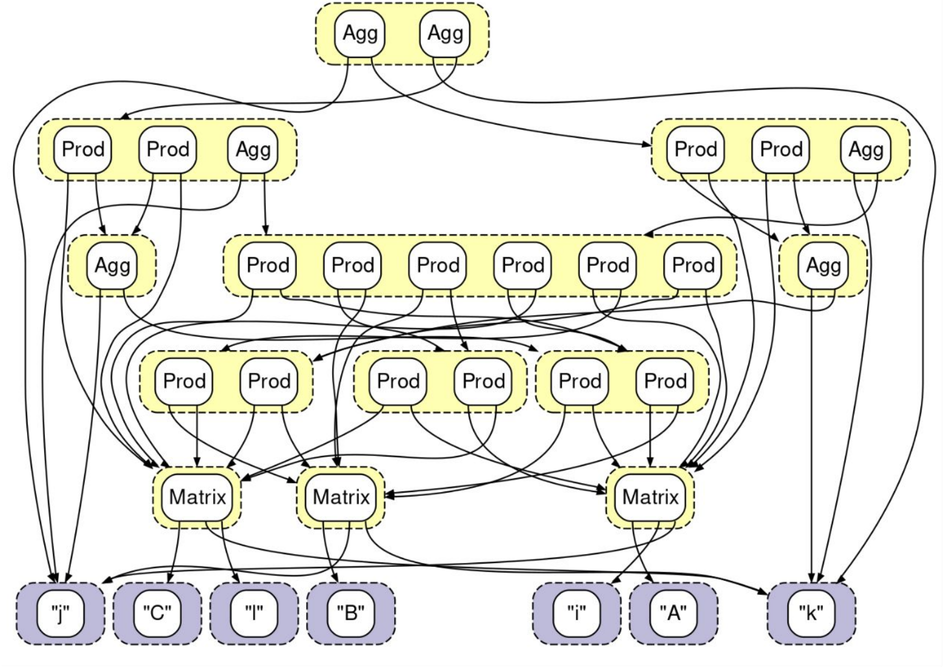


Example: $\Sigma_k A_{i,j}B_{j,k} = A_{i,j}(\Sigma_k B_{j,k})$

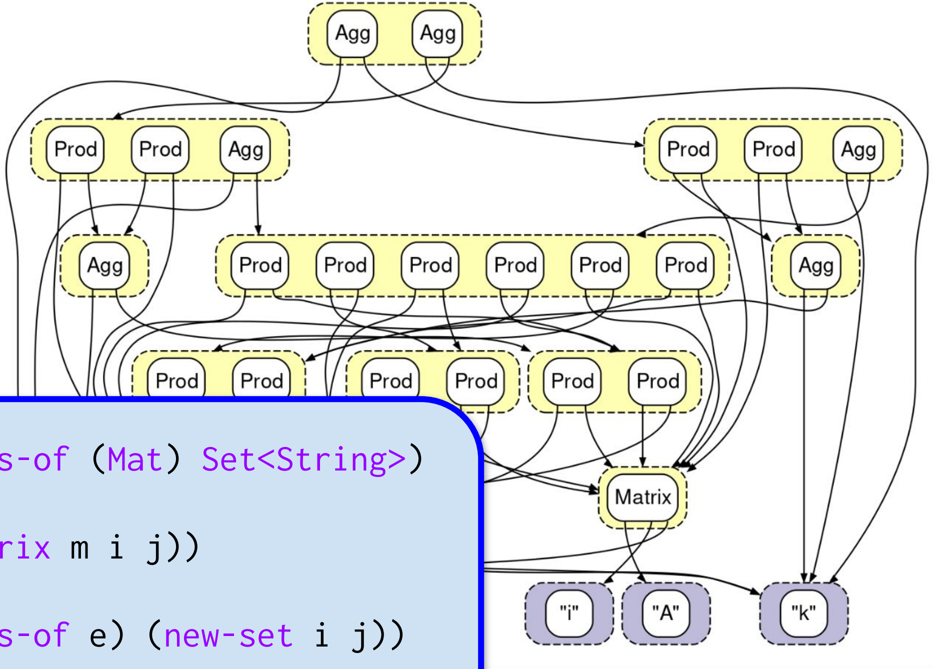
```
;; commutativity
(rewrite (Prod x y) (Prod y x))
;; associativity
(rewrite (Prod (Prod x y) z)
         (Prod x (Prod y z)))
;; commuting aggregation
(rewrite (Agg v1 (Agg v2 e))
         (Agg v2 (Agg v1 e)))
;; pushing down aggregation
(rewrite (Agg v (Prod x y))
         (Prod (Agg v x) y)
  :when ((∉ v (vars-of y))))
(rewrite (Agg v (Prod x y))
         (Prod x (Agg v y))
  :when ((∉ v (vars-of x)))
```



```
;; commutativity
(rewrite (Prod x y) (Prod y x))
;; associativity
(rewrite (Prod (Prod x y) z)
         (Prod x (Prod y z)))
;; commuting aggregation
(rewrite (Agg v1 (Agg v2 e))
         (Agg v2 (Agg v1 e)))
;; pushing down aggregation
(rewrite (Agg v (Prod x y))
         (Prod (Agg v x) y)
 :when ((∉ v (vars-of y))))
(rewrite (Agg v (Prod x y))
         (Prod x (Agg v y))
 :when ((∉ v (vars-of x)))
```



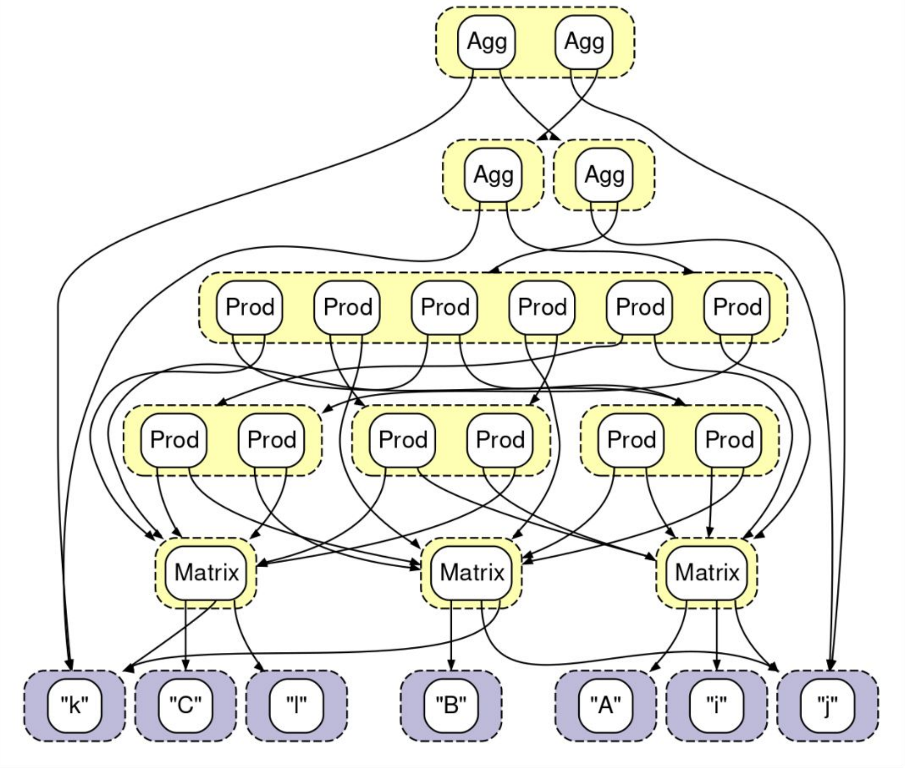
```
;; commutativity
(rewrite (Prod x y) (Prod y x))
;; associativity
(rewrite (Prod (Prod x y) z)
         (Prod x (Prod y z)))
;; commuting aggregation
(rewrite (Agg v1 (Agg v2 e))
         (Agg v2 (Agg v1 e)))
;; pushing down aggregation
(rewrite (Agg v (Prod x y))
         (Prod (Agg v x) y)
  :when ((∉ v (vars-of y)))
(rewrite (Agg v (Prod x y))
         (Prod x (Agg v y))
  :when ((∉ v (vars-of x)))
```



```
(function vars-of (Mat) Set<String>)
(rule (
    (= e (Matrix m i j))
) (
    (set (vars-of e) (new-set i j))
))
                    ...
```

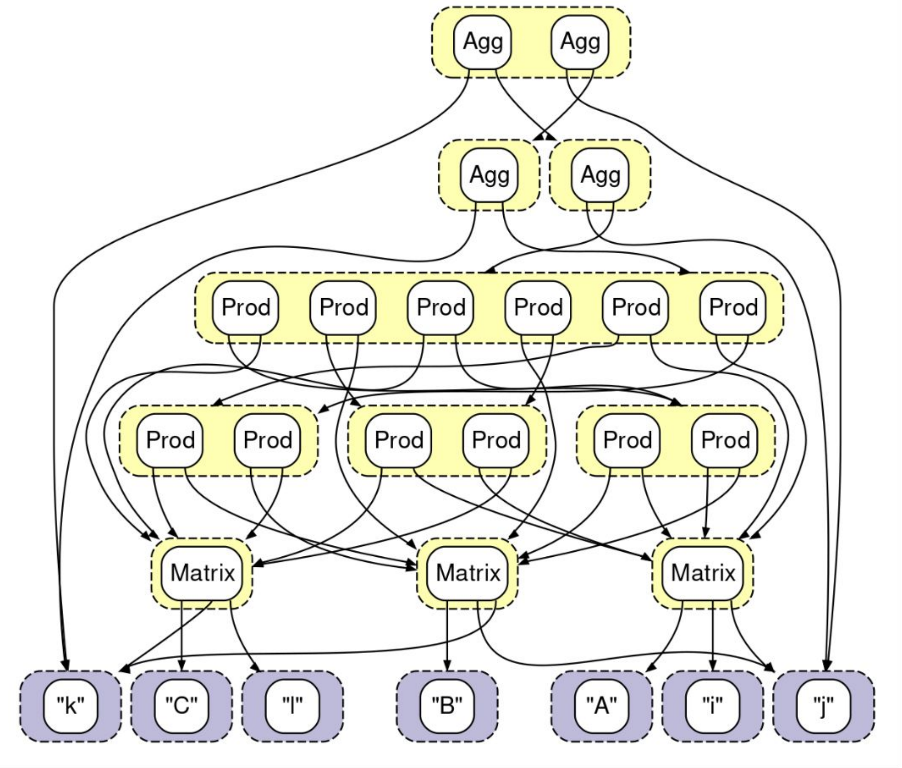
```
;; commutativity
(rewrite (Prod x y) (Prod y x))
;; associativity
(rewrite (Prod (Prod x y) z)
         (Prod x (Prod y z)))
;; commuting aggregation
(rewrite (Agg v1 (Agg v2 e))
         (Agg v2 (Agg v1 e)))
;; pushing down aggregation
(rewrite (Agg v (Prod x y))
         (Prod (Agg v x) y)
  :when ((∉ v (vars-of y))))
(rewrite (Agg v (Prod x y))
         (Prod x (Agg v y))
  :when ((∉ v (vars-of x)))
```

Rule-based Opt. ✓

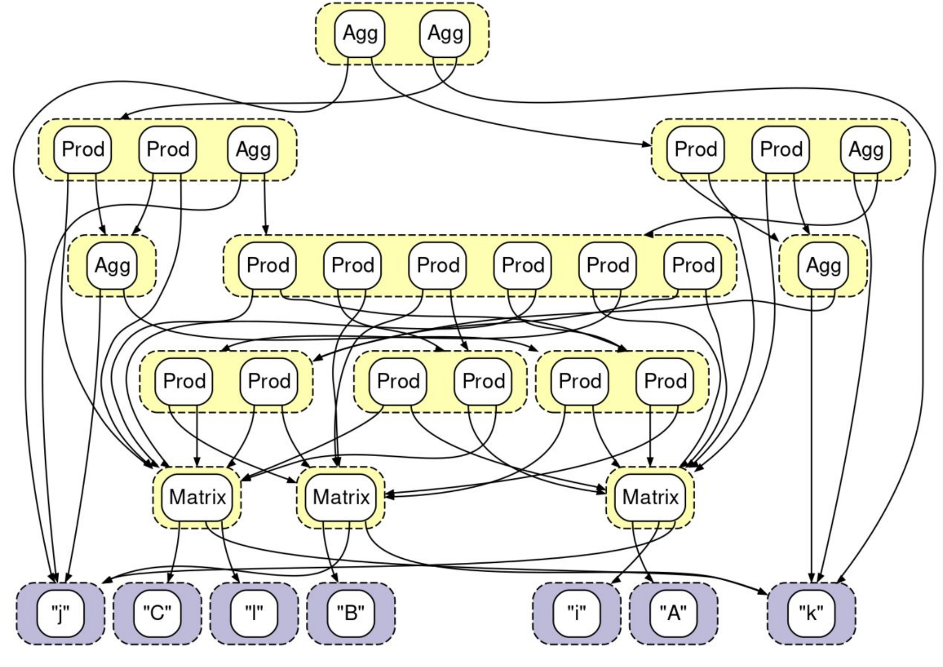


Cost-based Opt. ?

```
;; commutativity
(rewrite (Prod x y) (Prod y x))
;; associativity
(rewrite (Prod (Prod x y) z)
         (Prod x (Prod y z)))
;; commuting aggregation
(rewrite (Agg v1 (Agg v2 e))
         (Agg v2 (Agg v1 e)))
;; pushing down aggregation
(rewrite (Agg v (Prod x y))
         (Prod (Agg v x) y)
  :when ((∉ v (vars-of y))))
(rewrite (Agg v (Prod x y))
         (Prod x (Agg v y))
  :when ((∉ v (vars-of x)))
```

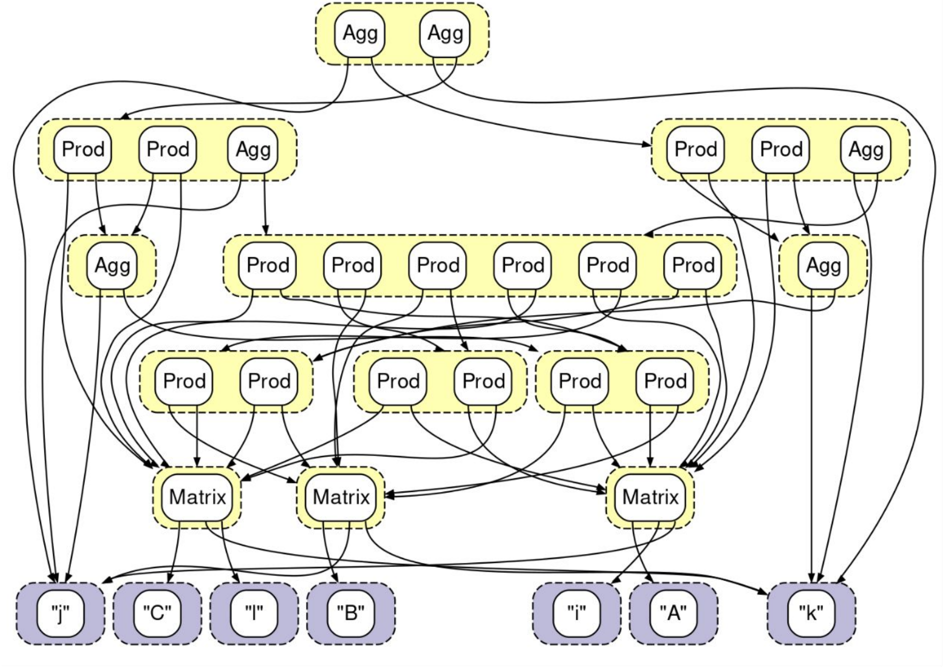


```
;; user provided dimension information
(function dim-of (String) i64)

;; estimate the size of a matrix expr
(function size-of (Mat) i64)
(rule (
    (= (vars-of e) vs)
) (
    (set (size-of e)
         (Π (map dim-of vs)))
))
```

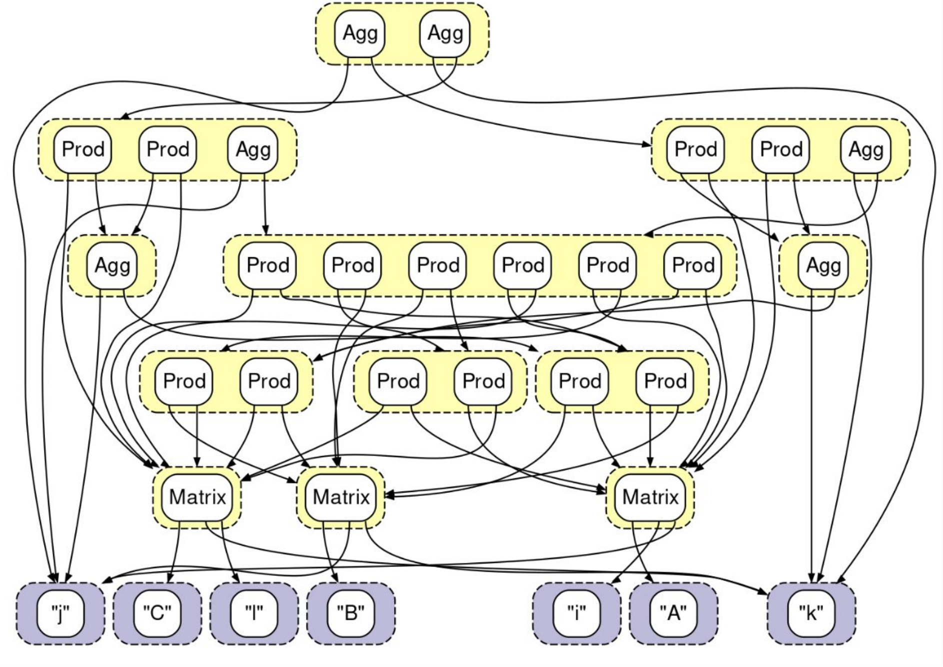


```
;; user provided dimension information
(function dim-of (String) i64)

;; estimate the size of a matrix expr
(function size-of (Mat) i64)
(rule (
    (= (vars-of e) vs)
) (
    (set (size-of e)
         (Π (map dim-of vs)))
))

;; set the cost of an expr as its size
(rule (
    (= (size-of (Prod e1 e2)) k)
) (
    (set-cost (Prod e1 e2) k)
))
```

…

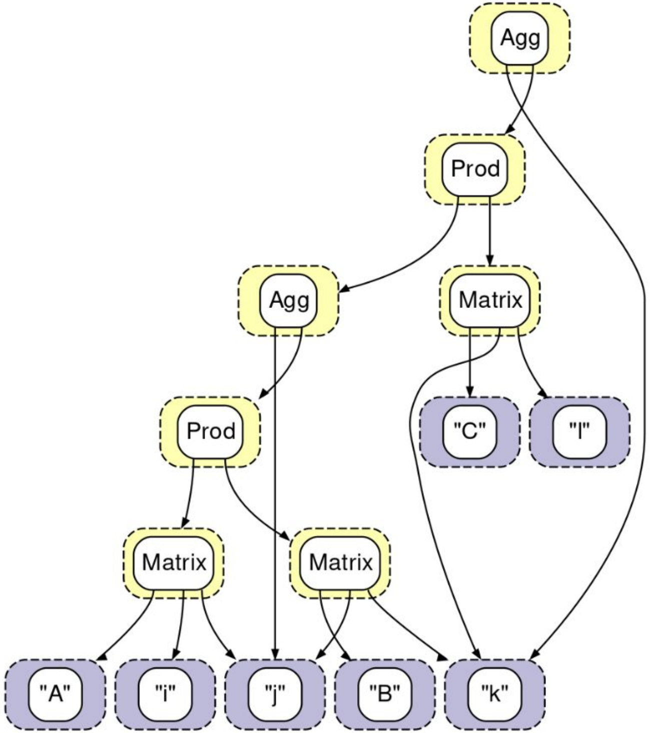


```
(set (dim-of "i") 256)
(set (dim-of "j") 64)
(set (dim-of "k") 16)
(set (dim-of "l") 256)
```



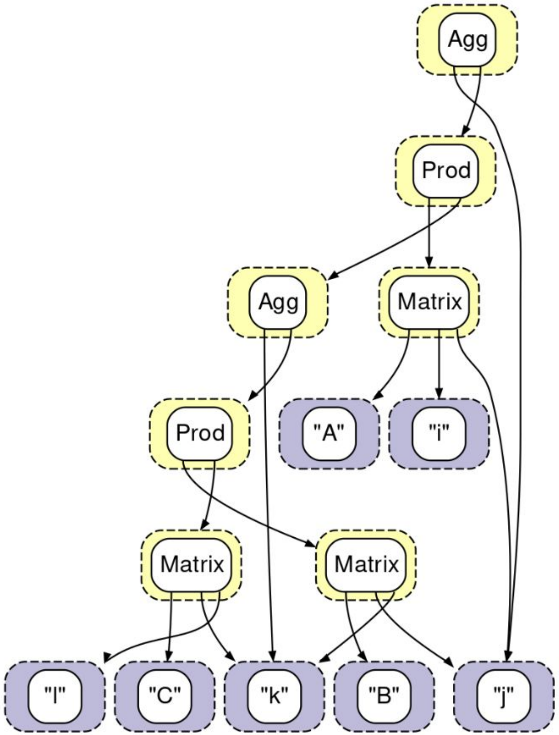
```
(set (dim-of "i") 256)
(set (dim-of "j") 64)
(set (dim-of "k") 16)
(set (dim-of "l") 256)

(extract ABC)
```

```
extracted with cost 1401867:
(Agg "k" (Prod
  (Agg "j" (Prod
    (Matrix "A" "i" "j")
    (Matrix "B" "j" "k")))
  (Matrix "C" "k" "l")))
```



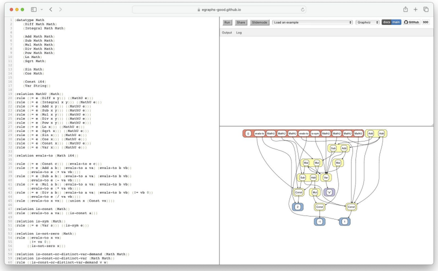
```
(set (dim-of "i") 256)
(set (dim-of "j") 64)
(set (dim-of "k") ~~16~~128)
(set (dim-of "l") 256)

(extract ABC)
```

```
extracted with cost 6430731:
(Agg "j" (Prod
  (Matrix "A" "i" "j")
  (Agg "k" (Prod
    (Matrix "B" "j" "k")
    (Matrix "C" "k" "l")))))
```

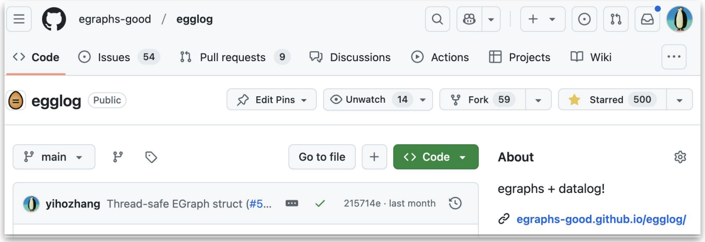


This talk

- EqSat: a promising approach to search-based program optimization
- EqSat ⊆ the Chase
- Cascades/Volcano ⊆ Equality Saturation
- EqSat unifies rule- and cost-based program optimization.



```
(datatype Math
    (Diff Math Math)
    (Integral Math Math)

    (Add Math Math)
    (Sub Math Math)
    (Mul Math Math)
    (Div Math Math)
    (Pow Math Math)
    (Ln Math)
    (Sqrt Math)

    (Sin Math)
    (Cos Math)

    (Const i64)
    (Var String))

(relation MathU (Math))
(rule ((= e (Diff x y))) ((MathU e)))
(rule ((= e (Integral x y))) ((MathU e)))
(rule ((= e (Add x y))) ((MathU e)))
(rule ((= e (Sub x y))) ((MathU e)))
(rule ((= e (Mul x y))) ((MathU e)))
(rule ((= e (Div x y))) ((MathU e)))
(rule ((= e (Pow x y))) ((MathU e)))
(rule ((= e (Ln x))) ((MathU e)))
(rule ((= e (Sqrt x))) ((MathU e)))
(rule ((= e (Sin x))) ((MathU e)))
(rule ((= e (Cos x))) ((MathU e)))
(rule ((= e (Const x))) ((MathU e)))
(rule ((= e (Var x))) ((MathU e)))

(relation evals-to (Math i64))

(rule ((= e (Const c))) ((evals-to e c)))
(rule ((= e (Add a b)) (evals-to a va) (evals-to b vb))
      ((evals-to e (+ va vb))))
(rule ((= e (Sub a b)) (evals-to a va) (evals-to b vb))
      ((evals-to e (- va vb))))
(rule ((= e (Mul a b)) (evals-to a va) (evals-to b vb))
      ((evals-to e (* va vb))))
(rule ((= e (Div a b)) (evals-to a va) (evals-to b vb) (!= vb 0))
      ((evals-to e (/ va vb))))
(rule ((evals-to x vx)) ((union x (Const vx))))

(relation is-const (Math))
(rule ((evals-to a va)) ((is-const a)))

(relation is-sym (Math))
(rule ((= e (Var s))) ((is-sym e)))

(relation is-not-zero (Math))
(rule ((evals-to x vx)
       (!= vx 0))
      ((is-not-zero x)))

(relation is-const-or-distinct-var-demand (Math Math))
(relation is-const-or-distinct-var (Math Math))
(rule ((is-const-or-distinct-var-demand v w)
```



yihozhang Thread-safe EGraph struct (#5... 215714e · last month

egraphs + datalog!

egraphs-good.github.io/egglog/

egraphs-good.github.io/egglog

github.com/egraphs-good/egglog