CSE599d: Advanced Query Processing

Lecture 8: Extensible Query Optimizers

Dan Suciu

University of Washington

Review

Σ-Algebras, $T(\Sigma, X)$

A signature is $\Sigma = \{f_1, f_2, \ldots, f_m\}$
each function symbol f_i and has an arity $n_i \geq 0$.

A Σ-algebra is $\boldsymbol{A} = (A, f_1^A, f_2^A, \ldots, f_m^A)$ where $f_i^A : A^{n_i} \rightarrow A$.

A Σ-term is an expression built from Σ and X.

The term algebra $T(\Sigma, X)$ = is the Σ-algebra of all Σ terms.
The "free Σ-algebra" generated by X.

Identities

An identity is a pair of terms (ℓ, r). We write $\ell \approx r$.

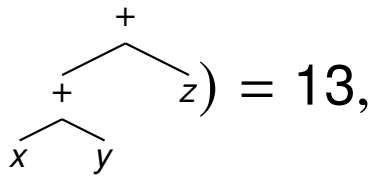
ℓ = the left-hand-side
r = the right-hand-side

Fix an algebra $\boldsymbol{A}$. A substitution is $\sigma : X \to A$.

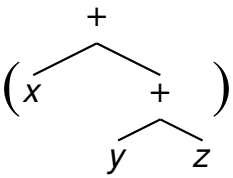
$\boldsymbol{A}$ satisfies the identity $\ell \approx r$ if $\sigma(\ell) = \sigma(r)$ for all σ

Notation: $\boldsymbol{A} \models \ell \approx r$

$\sigma : (x, y, z) \mapsto (5, 2, 6)$, then



$\sigma($ $) = 13,$



$\sigma($ $) = 13.$

Semantic Consequence

Fix a set of identities E.

An identity $s \approx t$ is a semantic consequence of E if:

For every Σ-algebra $\boldsymbol{A}$: $\boxed{\text{if } \boldsymbol{A} \models E \text{ then } \boldsymbol{A} \models s \approx t}$

We write $\approx_E$ for the semantic consequences of E.

It turns out that we can describe $\approx_E$ in terms of reductions.

The Reduction Relation

Review in class the reduction $\rightarrow_E$.

$\boxed{\overset{*}{\rightarrow_E}}$ and $\boxed{\overset{*}{\leftrightarrow}}$ extend $\rightarrow_E$ and are:

[1] σ is a substitution $\sigma : X \rightarrow T(\Sigma, X)$.

The Reduction Relation

Review in class the reduction $\rightarrow_E$.

$\boxed{\overset{*}{\rightarrow_E}}$ and $\boxed{\overset{*}{\leftrightarrow}}$ extend $\rightarrow_E$ and are:

Reflexive $t \overset{*}{\rightarrow}_E t$; $\overset{*}{\leftrightarrow}$ is also Symmetric

Transitive $s \overset{*}{\rightarrow}_E t, t \overset{*}{\rightarrow}_E u$ implies $s \overset{*}{\rightarrow}_E u$.

[1] σ is a substitution $\sigma : X \rightarrow T(\Sigma, X)$.

The Reduction Relation

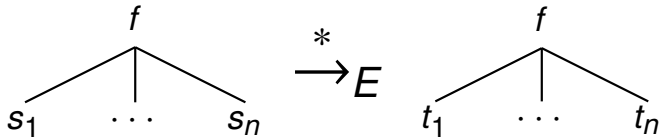
Review in class the reduction $\rightarrow_E$.

$\boxed{\overset{*}{\rightarrow_E}}$ and $\boxed{\overset{*}{\leftrightarrow}}$ extend $\rightarrow_E$ and are:

Reflexive $t \overset{*}{\rightarrow_E} t$; $\overset{*}{\leftrightarrow}$ is also Symmetric

Transitive $s \overset{*}{\rightarrow_E} t, t \overset{*}{\rightarrow_E} u$ implies $s \overset{*}{\rightarrow_E} u$.

Congruence If $s_1 \overset{*}{\rightarrow_E} t_1, \ldots, s_n \overset{*}{\rightarrow_E} t_n$ then:



[1] σ is a substitution $\sigma : X \rightarrow T(\Sigma, X)$.

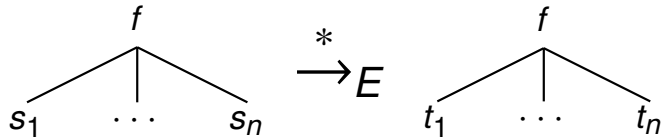
The Reduction Relation

Review in class the reduction $\rightarrow_E$.

$\boxed{\overset{*}{\rightarrow_E}}$ and $\boxed{\overset{*}{\leftrightarrow}}$ extend $\rightarrow_E$ and are:

Reflexive $t \overset{*}{\rightarrow_E} t$; $\overset{*}{\leftrightarrow}$ is also Symmetric

Transitive $s \overset{*}{\rightarrow_E} t, t \overset{*}{\rightarrow_E} u$ implies $s \overset{*}{\rightarrow_E} u$.

Congruence If $s_1 \overset{*}{\rightarrow_E} t_1, \ldots, s_n \overset{*}{\rightarrow_E} t_n$ then: 

Closed under substitutions[1] If $s \rightarrow t \in E$, then $\sigma(s) \overset{*}{\rightarrow_E} \sigma(t)$ for all σ

Theorem [Birkoff] $\approx_E$ is the same as $\overset{*}{\leftrightarrow_E}$

[1] σ is a substitution $\sigma : X \rightarrow T(\Sigma, X)$.

The Word Problem

Fix Σ, E.

The Word Problem: given s, t, decide whether $s \approx_E t$

Decidable cases: semigroup, monoids, groups, lattices

Undecidable cases: Matijasevič's strange identities; combinatory logic.

A Special Case

Theorem when all terms in E are grounded (i.e. no variables), then the ground word problem (i.e. s, t are also grounded) is decidable.

Relational Algebra v.s. a Σ-algebra

$\Sigma = \{\sigma, \Pi, \bowtie, \cup, -\}$. However:

- Some operations are undefined: $R \cup S$ when R, S have different schemas.
- Some operations have parameters: $\sigma_p(R)$ where p is some predicate.
- Template identities: $R \bowtie_{A=B} S \approx S \bowtie_{B=A} R$ for every choice of A, B.

Could fix the theory. Volcano/Cascades just ignore these rough spots.

No Complete Identities for Relational Algebra!

Query equivalence, $Q \equiv Q'$, means: $Q(D) = Q'(D)$ for any database D.

Fact There is no r.e.[2] E for which $\approx_E$ coincides with $\equiv$.

Because:

- $Q \equiv Q'$ is undecidable
- $\approx_E$ is r.e.
- $\equiv$ is co-r.e.
- If $\approx_E$ conicides with $\equiv$, then they were decidable

[2] Recursively enumerable (e.g. templates).

E-Graphs

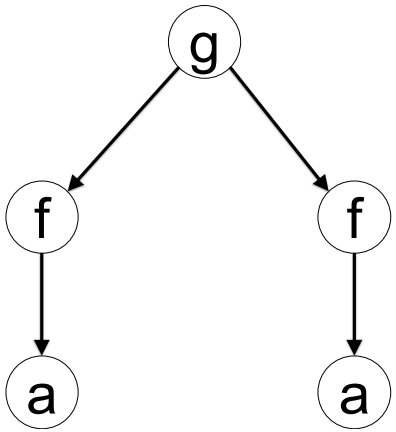
Overview

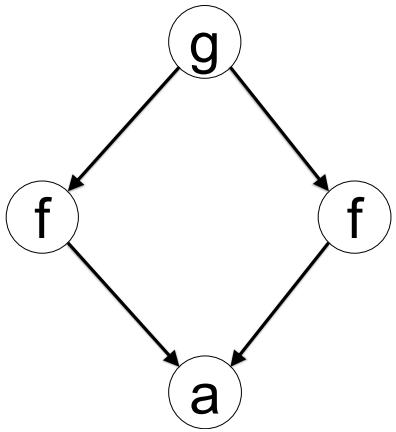
E-graphs are keeping all intermediate steps of a rewrite sequence $s \to s_1 \to s_2 \to \cdots$ in a compact data structure.

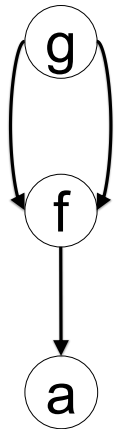
This part is based on the book [Baader and Nipkow, 1999] and the egg paper [Willsey et al., 2021].

Term Graph

Term Graph is a DAG sharing common subexpressions:





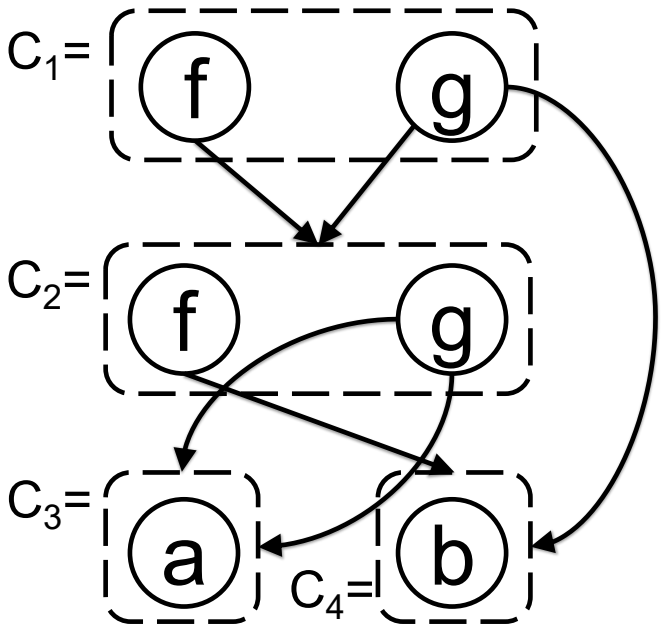


E-Graph

An E-graph G consists of:

- A set of E-classes $c_1, c_2, \ldots,$
- $\ldots$ where each E-class c is a set of E-nodes $c = \{v_1, v_2, \ldots\},$
- $\ldots$ and each e-node is a pair $(f, (c_{i_1}, \ldots, c_{i_k}))$ where $f \in \Sigma$, $c_{i_1}, \ldots, c_{i_k}$ are E-classes, and $k =$ the arity of f.

E-Graphs



Each class c represents a set of terms:

$\text{terms}(c_3) = a,$

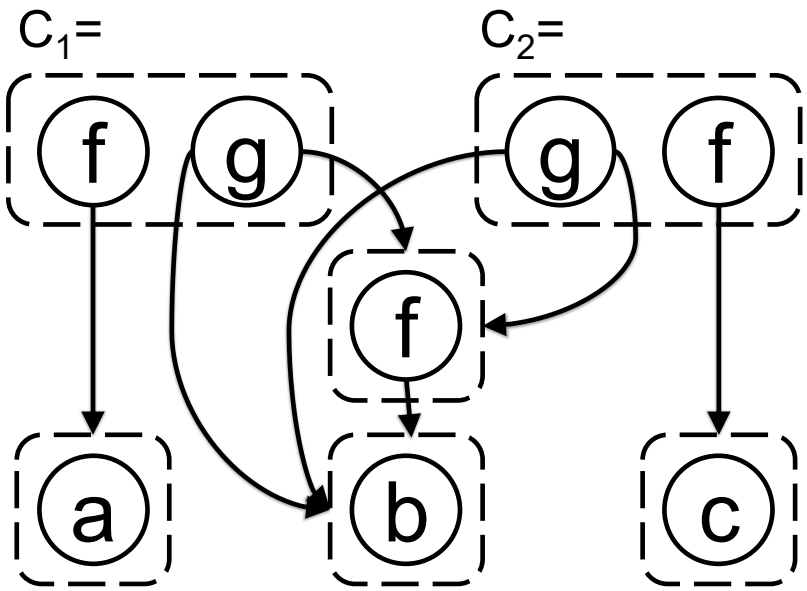
$\text{terms}(c_4) = b$

$\text{terms}(c_2) = \{f(b), g(a,a)\}$

$$\text{terms}(c_1) = \{f(f(b)), f(g(a,a)), g(f(b),b), g(g(a,a),b)\}$$

$\text{terms}(G)$ = the set of all terms represented by all E-classes.

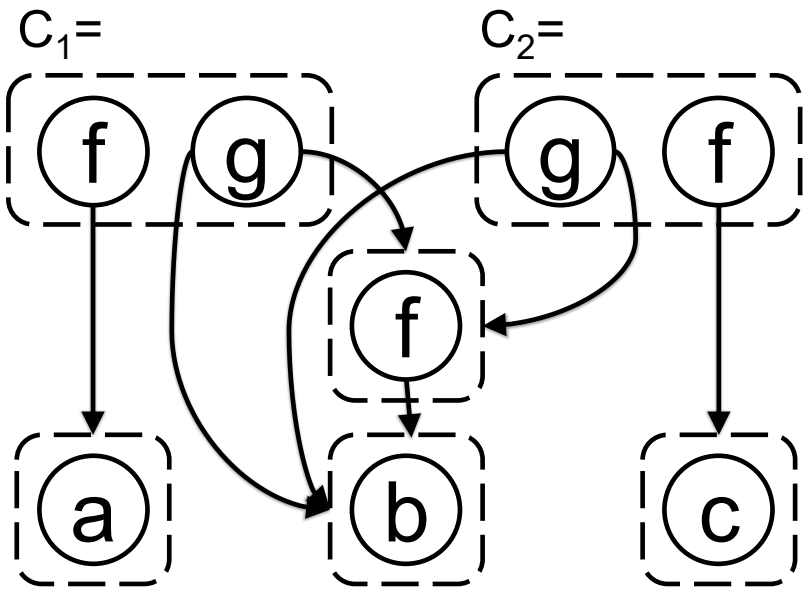
Congruence Closure



terms(c_1) = ????

terms(c_2) = ????

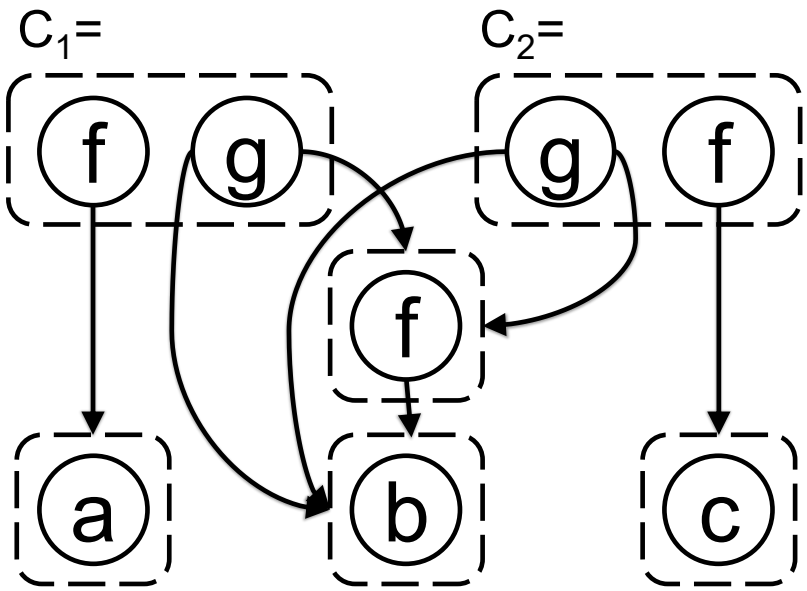
Congruence Closure



$\text{terms}(c_1) = \{f(a), g(b, f(b))\}$

$\text{terms}(c_2) =$????

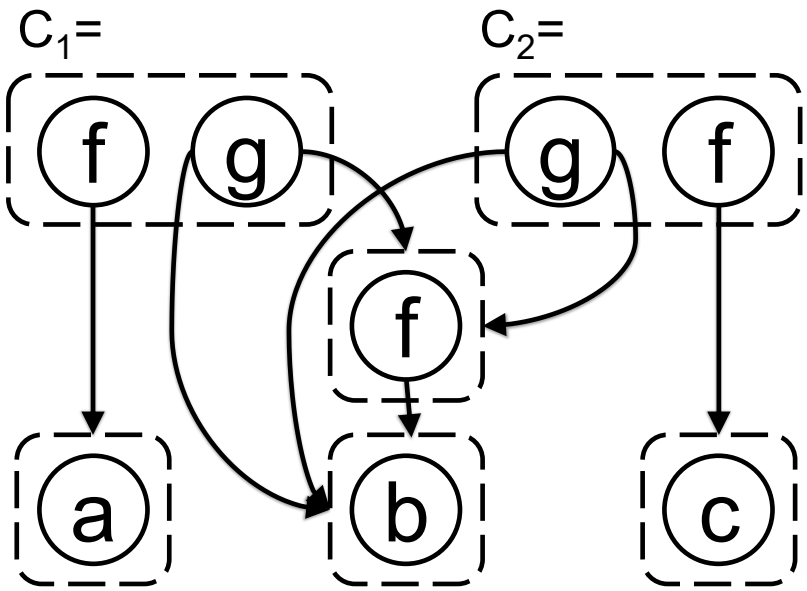
Congruence Closure



$$\text{terms}(c_1) = \{f(a), g(b, f(b))\}$$

$$\text{terms}(c_2) = \{g(b, f(b)), f(c)\}$$

Congruence Closure

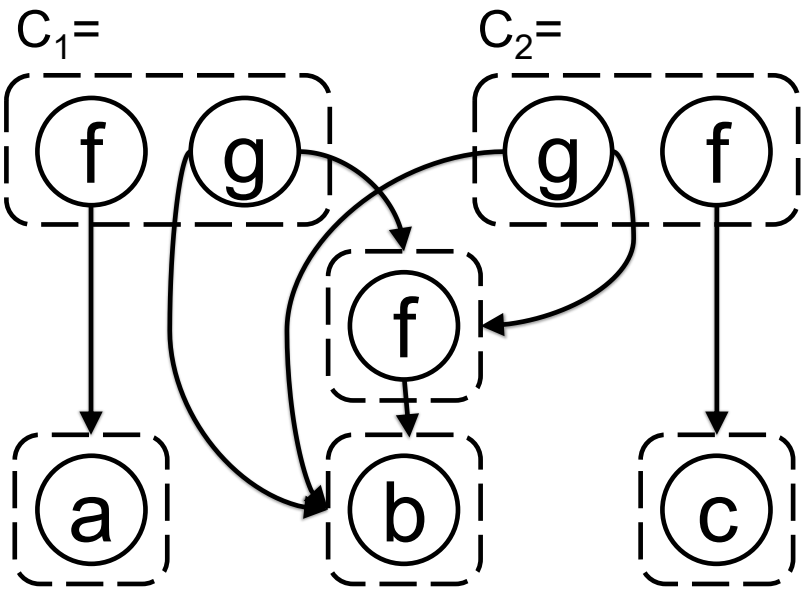


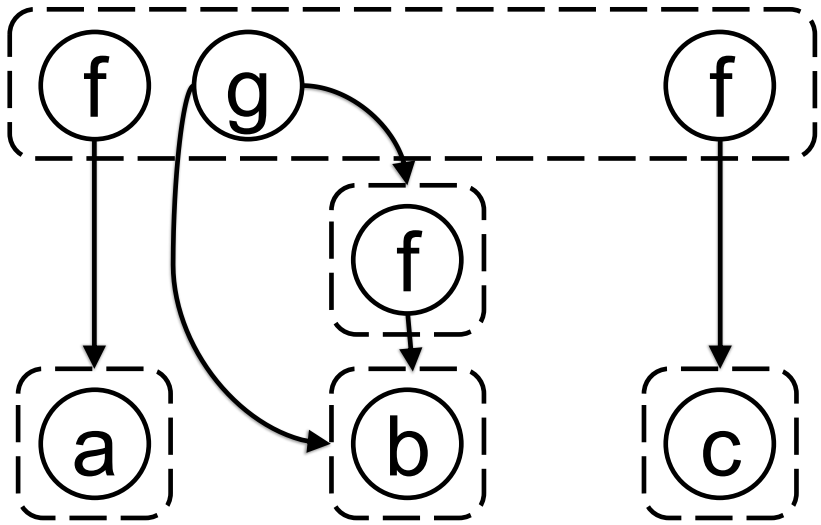
$\text{terms}(c_1) = \{f(a), g(b, f(b))\}$

$\text{terms}(c_2) = \{g(b, f(b)), f(c)\}$

Common term $g(b, f(b))$

Congruence Closure





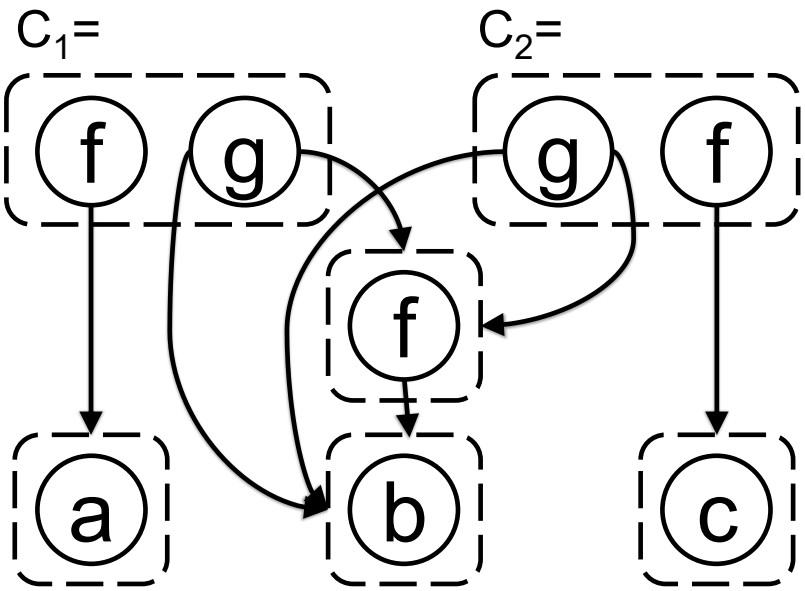
$\text{terms}(c_1) = \{f(a), g(b, f(b))\}$

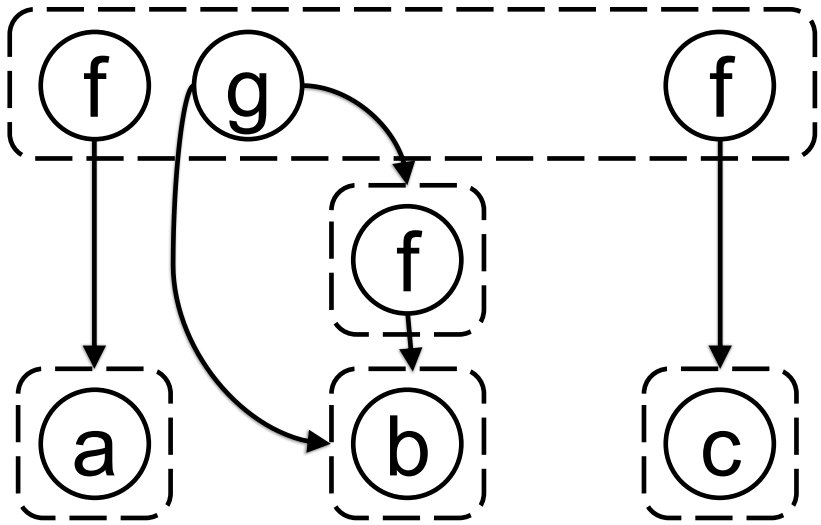
$\text{terms}(c_2) = \{g(b, f(b)), f(c)\}$

Common term $g(b, f(b))$

Congruence closure of the E-graph

Congruence Closure





$\text{terms}(c_1) = \{f(a), g(b, f(b))\}$

$\text{terms}(c_2) = \{g(b, f(b)), f(c)\}$

Common term $g(b, f(b))$

Congruence closure of the E-graph

We inferred: $f(a) = f(c)$

Congruence Closure

An E-graph defines the following relation on terms(G):

$$s \sim_G t \text{ if exists E-class } c \text{ such that } s, t \in \text{terms}(c)$$

Congruence Closure

An E-graph defines the following relation on terms(G):

$$\boxed{s \sim_G t \text{ if exists E-class } c \text{ such that } s, t \in \text{terms}(c)}$$

We say that G is closed under congruence if $\sim_G$ is a congruence on terms(G):

- $\sim_G$ is an equivalence relation, and
- if $f(s_1, \ldots, s_n), f(t_1, \ldots, t_n) \in \text{terms}(G)$
  and $s_1 \sim_G t_1, s_2 \sim_G t_2, \ldots,$ then $f(s_1, \ldots, s_n) \sim_G f(t_1, \ldots, t_n)$.

Congruence Closure

An E-graph defines the following relation on terms(G):

$$\boxed{s \sim_G t \text{ if exists E-class } c \text{ such that } s, t \in \text{terms}(c)}$$

We say that G is closed under congruence if $\sim_G$ is a congruence on terms(G):

- $\sim_G$ is an equivalence relation, and
- if $f(s_1, \ldots, s_n), f(t_1, \ldots, t_n) \in \text{terms}(G)$
  and $s_1 \sim_G t_1, s_2 \sim_G t_2, \ldots,$ then $f(s_1, \ldots, s_n) \sim_G f(t_1, \ldots, t_n)$.

Congruence closure algorithm: modifies G to ensure that $\sim_G$ is a congruence.

Approaches to the Word Problem

Input: E, s, t Question: is $s \approx_E t$?

Approaches to the Word Problem

Input: E, s, t Question: is $s \approx_E t$?

TRS approach

$$\begin{array}{cccccccc} s & \to_E & s_1 & \to_E & s_2 & \to_E & \cdots \\ t & \to_E & t_1 & \to_E & t_2 & \to_E & \cdots \end{array}$$

Common term?
Then $s \approx_E t$.

Approaches to the Word Problem

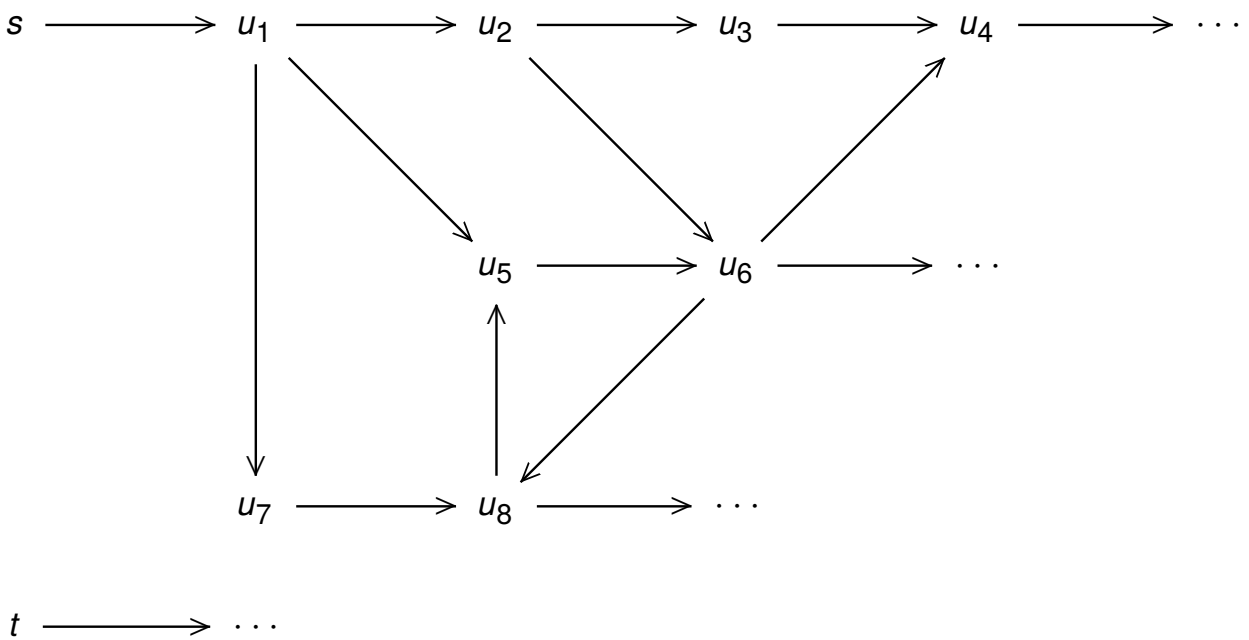
Input: E, s, t Question: is $s \approx_E t$?

TRS approach

$$\begin{array}{cccccccc} s & \to_E & s_1 & \to_E & s_2 & \to_E & \cdots \\ t & \to_E & t_1 & \to_E & t_2 & \to_E & \cdots \end{array}$$

Common term?
Then $s \approx_E t$.

E-graph approach encode many derivations in the E-graph:



Approaches to the Word Problem

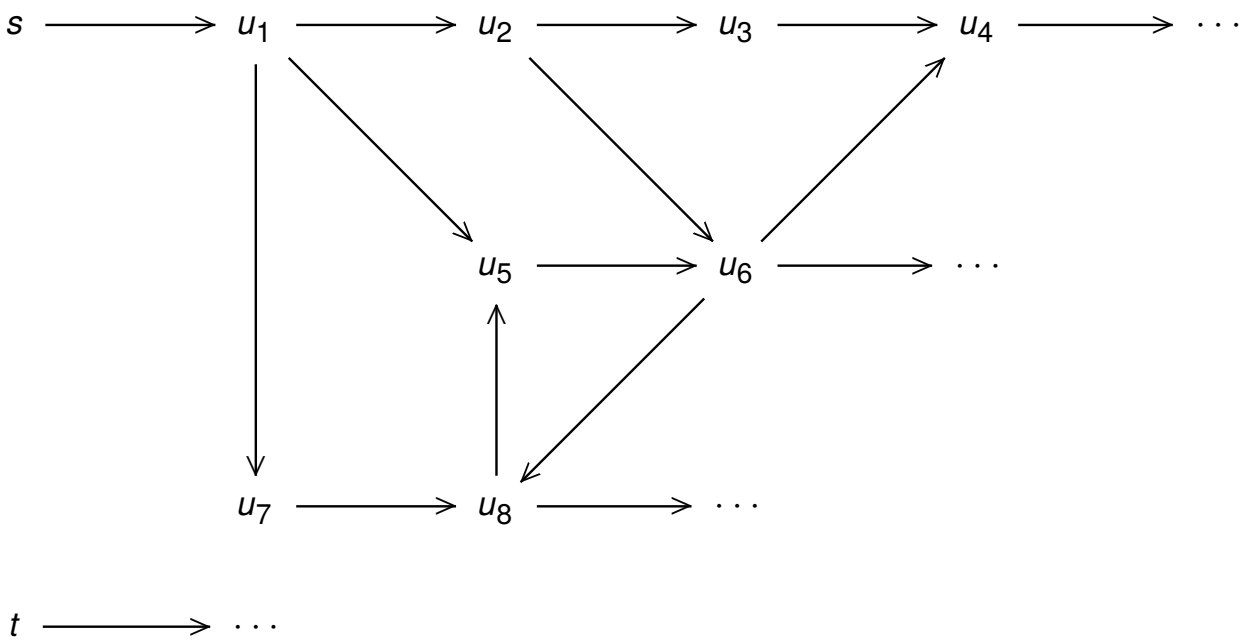
Input: E, s, t Question: is $s \approx_E t$?

TRS approach

$$\begin{array}{cccccccc} s & \to_E & s_1 & \to_E & s_2 & \to_E & \cdots \\ t & \to_E & t_1 & \to_E & t_2 & \to_E & \cdots \end{array}$$

Common term?
Then $s \approx_E t$.

E-graph approach encode many derivations in the E-graph:



Both approaches assume that E is directed.

Solving the Word Problem with E-graphs

The Equality Saturation Algorithm To check $s \equiv_E t$:

Initialize the E-graph with s, t.

Solving the Word Problem with E-graphs

The Equality Saturation Algorithm To check $s \equiv_E t$:

Initialize the E-graph with s, t.

- Find $(\ell, r) \in E$ such that ℓ "matches" the E-graph.
  Meaning: $\exists$ class c, substitution σ, s.t. $\sigma(\ell) \in \text{terms}(c)$.

Solving the Word Problem with E-graphs

The Equality Saturation Algorithm To check $s \equiv_E t$:

Initialize the E-graph with s, t.

- Find $(\ell, r) \in E$ such that ℓ "matches" the E-graph.
  Meaning: $\exists$ class c, substitution σ, s.t. $\sigma(\ell) \in \text{terms}(c)$.
- Add $\sigma(r)$ to G, $c := c \cup \{\text{root}(\sigma(r))\}$.

Solving the Word Problem with E-graphs

The Equality Saturation Algorithm To check $s \equiv_E t$:

Initialize the E-graph with s, t.

- Find $(\ell, r) \in E$ such that ℓ "matches" the E-graph.
  Meaning: $\exists$ class c, substitution σ, s.t. $\sigma(\ell) \in \text{terms}(c)$.
- Add $\sigma(r)$ to G, $c := c \cup \{\text{root}(\sigma(r))\}$.
- Perform congruence closure
- Repeat.

Solving the Word Problem with E-graphs

The Equality Saturation Algorithm To check $s \equiv_E t$:

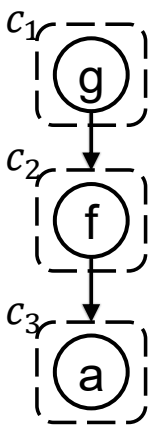
Initialize the E-graph with s, t.

- Find $(\ell, r) \in E$ such that ℓ "matches" the E-graph.
  Meaning: $\exists$ class c, substitution σ, s.t. $\sigma(\ell) \in \text{terms}(c)$.
- Add $\sigma(r)$ to G, $c := c \cup \{\text{root}(\sigma(r))\}$.
- Perform congruence closure
- Repeat.

When (and if!) it terminates, check $\boxed{s \sim_G t}$

Example from [Baader and Nipkow, 1999]

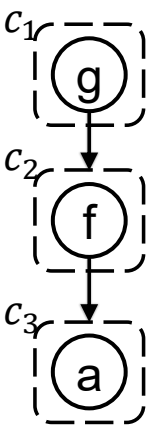
$E = \boxed{g(x) \approx f(f(x))}$ Check $\boxed{g(f(a)) \approx_E f(g(a))}$



Example from [Baader and Nipkow, 1999]

$E = \boxed{g(x) \approx f(f(x))}$

Check $\boxed{g(f(a)) \approx_E f(g(a))}$



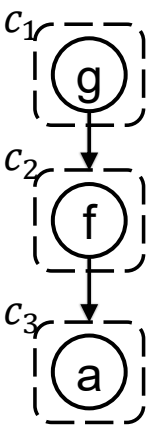
$g(x) \rightarrow f(f(x))$

$\sigma(x) = c_2$

Insert $f(f(c_2))$

Example from [Baader and Nipkow, 1999]

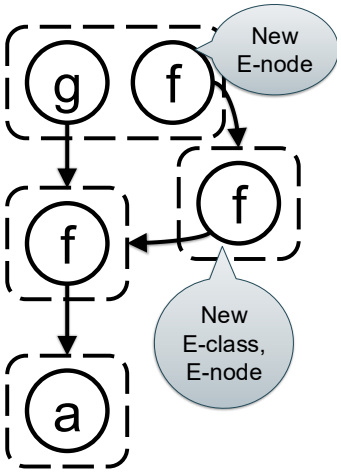
$E = \boxed{g(x) \approx f(f(x))}$ Check $\boxed{g(f(a)) \approx_E f(g(a))}$



$g(x) \rightarrow f(f(x))$

$\sigma(x) = c_2$

Insert $f(f(c_2))$

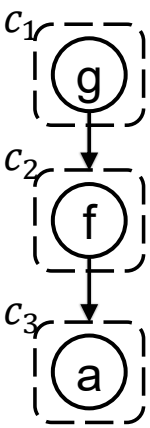


Example from [Baader and Nipkow, 1999]

$E = \boxed{g(x) \approx f(f(x))}$

Check $\boxed{g(f(a)) \approx_E f(g(a))}$

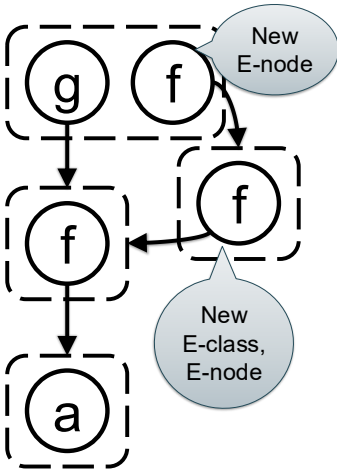
Stuck! Need to close E under symmetry



$g(x) \rightarrow f(f(x))$

$\sigma(x) = c_2$

Insert $f(f(c_2))$

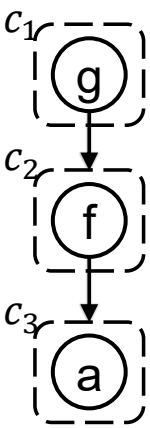


Example from [Baader and Nipkow, 1999]

$E = \boxed{\begin{array}{l} g(x) \approx f(f(x)) \\ f(f(x)) \approx g(x) \end{array}}$

Check $\boxed{g(f(a)) \approx_E f(g(a))}$

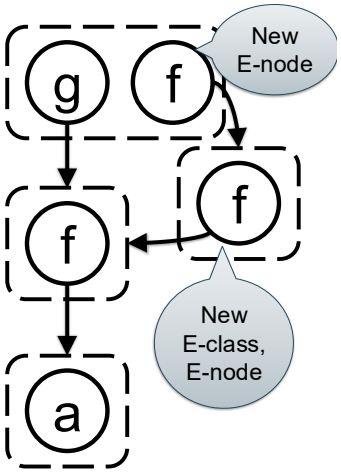
Stuck! Need to close E under symmetry



$g(x) \rightarrow f(f(x))$

$\sigma(x) = c_2$

Insert $f(f(c_2))$

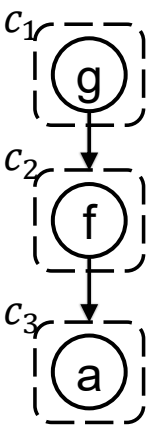


Example from [Baader and Nipkow, 1999]

$$E = \boxed{\begin{array}{l} g(x) \approx f(f(x)) \\ f(f(x)) \approx g(x) \end{array}}$$

Check $\boxed{g(f(a)) \approx_E f(g(a))}$

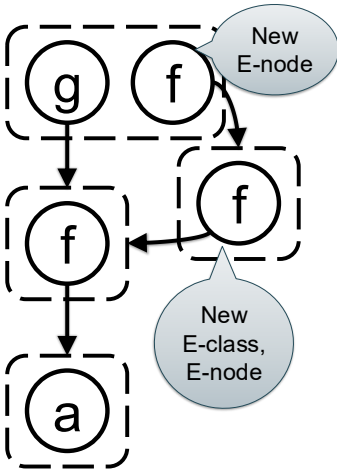
Stuck! Need to close E under symmetry



$g(x) \rightarrow f(f(x))$

$\sigma(x) = c_2$

Insert $f(f(c_2))$



$f(f(x)) \rightarrow g(x)$

$\sigma(x) = c_3$

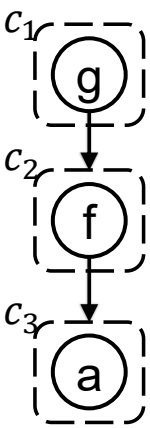
Insert $g(c_3)$

Example from [Baader and Nipkow, 1999]

$$E = \boxed{\begin{array}{l} g(x) \approx f(f(x)) \\ f(f(x)) \approx g(x) \end{array}}$$

Check $\boxed{g(f(a)) \approx_E f(g(a))}$

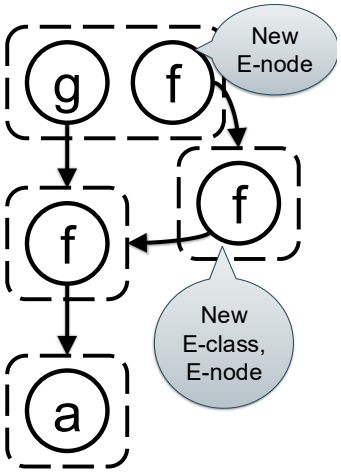
Stuck! Need to close E under symmetry



$g(x) \rightarrow f(f(x))$

$\sigma(x) = c_2$

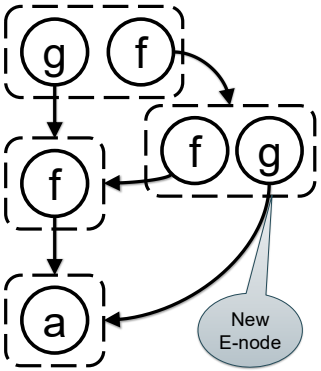
Insert $f(f(c_2))$



$f(f(x)) \rightarrow g(x)$

$\sigma(x) = c_3$

Insert $g(c_3)$

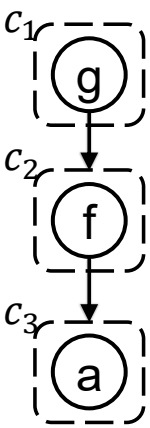


Example from [Baader and Nipkow, 1999]

$E = \boxed{\begin{array}{l} g(x) \approx f(f(x)) \\ f(f(x)) \approx g(x) \end{array}}$

Check $\boxed{g(f(a)) \approx_E f(g(a))}$

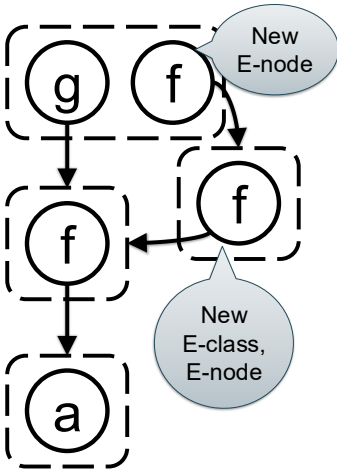
Stuck! Need to close E under symmetry



$g(x) \rightarrow f(f(x))$

$\sigma(x) = c_2$

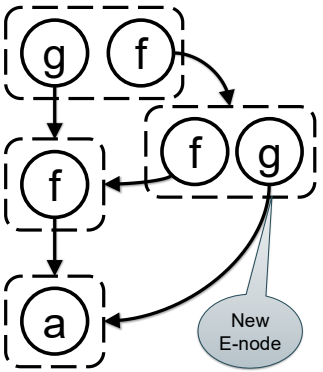
Insert $f(f(c_2))$



$f(f(x)) \rightarrow g(x)$

$\sigma(x) = c_3$

Insert $g(c_3)$



$g(f(a)) \in \text{terms}(c_1)$ and also $f(g(a)) \in \text{terms}(c_1)$

Hence $g(f(a)) \sim_G f(g(a))$ which implies $g(f(a)) \approx_E f(g(a))$.

Discussion (1/2)

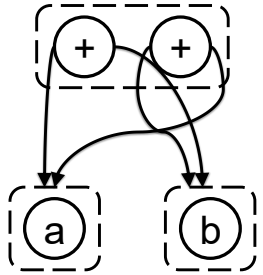
The Equality Saturation Algorithm ensures that $s \sim_G t$ implies $s \approx_E t$.

Converse fails for multiple reasons:

- E is treated as asymmetrically and may fail to trigger.
- $\sim_G$ is an equivalence relation only on terms(G).
  Called Partial Equivalence Relation, PER; also partial congruence.

Discussion (2/2)

E-graphs can avoid some non-terminating reductions:,
E.g. commutativity:



Non-termination can still happen, e.g. $g(x) \approx g(f(x))$:
Leads to $g(a) \approx g(f(a)) \approx g(f(f(a))) \approx \cdots$

Direction of the identities in E may affect termination: e.g. $g(f(x)) \approx g(x)$.

No good theory exists for understanding termination.

Special Case: Grounded Identities

Theorem when all terms in E are grounded (i.e. no variables), then the ground word problem (i.e. s, t are also grounded) is decidable.

Special Case: Grounded Identities

Theorem when all terms in E are grounded (i.e. no variables), then the ground word problem (i.e. s, t are also grounded) is decidable.

Proof Initialize E-graph with s, t, and all sub-terms of terms E.

Make E symmetric: $\ell \to_E r$ and $r \to_E \ell$.

Special Case: Grounded Identities

Theorem when all terms in E are grounded (i.e. no variables), then the ground word problem (i.e. s, t are also grounded) is decidable.

Proof Initialize E-graph with s, t, and all sub-terms of terms E.

Make E symmetric: $\ell \rightarrow_E r$ and $r \rightarrow_E \ell$.

Equality saturation never needs to insert new E-nodes in G

Hence the congruence closure algorithm terminates
and $\sim_G$ is the same as $\overset{*}{\leftrightarrow}_E$ on terms(G) (because E is symmetric).

Example from [Baader and Nipkow, 1999]

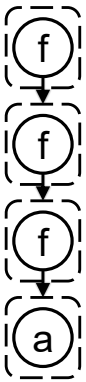
$E = \boxed{\begin{array}{l} f(f(a)) \approx a \\ f(f(f(a))) \approx a \end{array}}$

Check if $\boxed{f(a) \approx_E a}$

Example from [Baader and Nipkow, 1999]

$E = \boxed{\begin{array}{l} f(f(a)) \approx a \\ f(f(f(a))) \approx a \end{array}}$

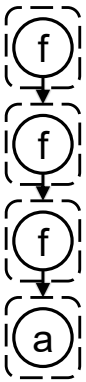
Check if $\boxed{f(a) \approx_E a}$



Example from [Baader and Nipkow, 1999]

$E = \boxed{\begin{array}{l} f(f(a)) \approx a \\ f(f(f(a))) \approx a \end{array}}$

Check if $\boxed{f(a) \approx_E a}$

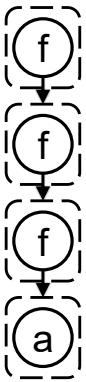


$$f(f(f(a))) \rightarrow a$$

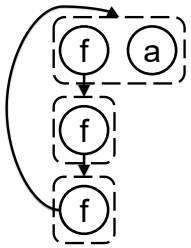
Example from [Baader and Nipkow, 1999]

$E = \boxed{\begin{array}{l} f(f(a)) \approx a \\ f(f(f(a))) \approx a \end{array}}$

Check if $\boxed{f(a) \approx_E a}$



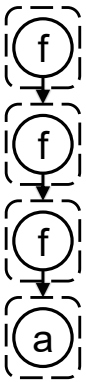
$f(f(f(a))) \rightarrow a$



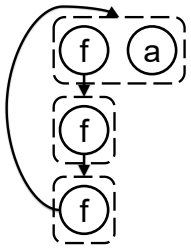
Example from [Baader and Nipkow, 1999]

$E = \boxed{\begin{array}{l} f(f(a)) \approx a \\ f(f(f(a))) \approx a \end{array}}$

Check if $\boxed{f(a) \approx_E a}$



$f(f(f(a))) \rightarrow a$

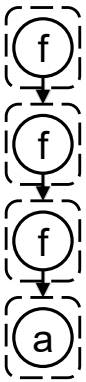


$f(f(a)) \rightarrow a$

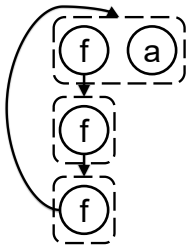
Example from [Baader and Nipkow, 1999]

$E = \boxed{\begin{array}{l} f(f(a)) \approx a \\ f(f(f(a))) \approx a \end{array}}$

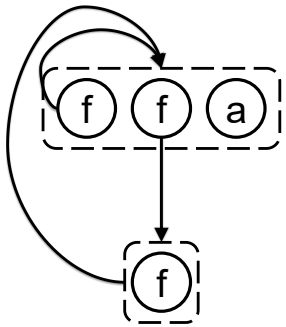
Check if $\boxed{f(a) \approx_E a}$



$f(f(f(a))) \rightarrow a$



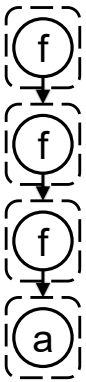
$f(f(a)) \rightarrow a$



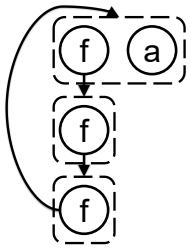
Example from [Baader and Nipkow, 1999]

$E = \boxed{\begin{array}{l} f(f(a)) \approx a \\ f(f(f(a))) \approx a \end{array}}$

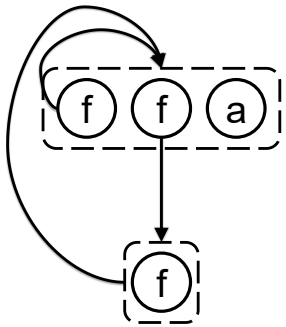
Check if $\boxed{f(a) \approx_E a}$



$f(f(f(a))) \rightarrow a$



$f(f(a)) \rightarrow a$

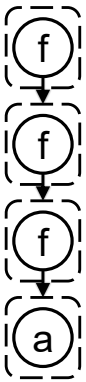


$f(f(f(a))) \rightarrow a$

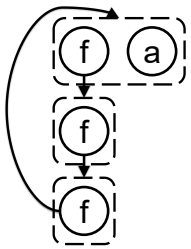
Example from [Baader and Nipkow, 1999]

$E = \boxed{\begin{array}{l} f(f(a)) \approx a \\ f(f(f(a))) \approx a \end{array}}$

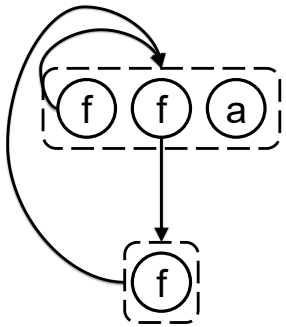
Check if $\boxed{f(a) \approx_E a}$



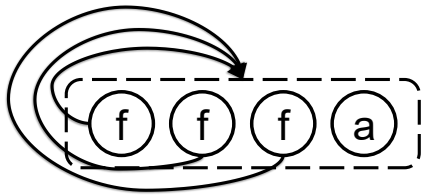
$f(f(f(a))) \rightarrow a$



$f(f(a)) \rightarrow a$



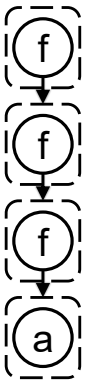
$f(f(f(a))) \rightarrow a$



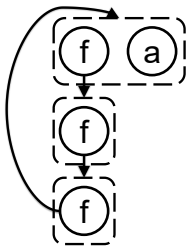
Example from [Baader and Nipkow, 1999]

$E = \boxed{\begin{array}{l} f(f(a)) \approx a \\ f(f(f(a))) \approx a \end{array}}$

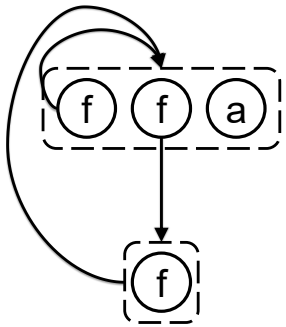
Check if $\boxed{f(a) \approx_E a}$



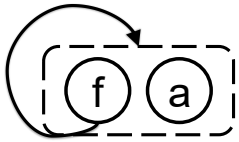
$f(f(f(a))) \rightarrow a$



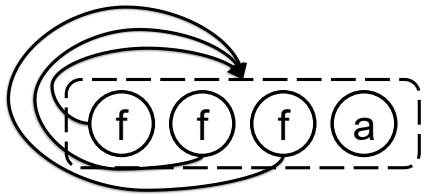
$f(f(a)) \rightarrow a$



$f(f(f(a))) \rightarrow a$



$\Leftarrow$



Volcano/Cascades

Overview

Lecture mostly based on [Ding et al., 2024]

Volcano:

- Introduced in [Graefe, 1994]; more details [McKenna, 1993]
- Logical rules: equality saturation; physical rules: branch-and-bound.

Cascades

- Introduced in [Graefe, 1995]; more details [Xu, 1998]
- Combines the two phases of Volcano.
- Adoption: SQLServer, Calcite, CockroachDB

Terminology (1/2)

Logical Property of an expression: e.g. estimated cardinality

Physical Property of an expression: interesting sort order, partition.

Logical Operators e.g. $\bowtie$, σ, Γ, $\rtimes$ ...

Physical Operators e.g. `HashJoin`, `MergeJoin`, `IndexScan`, ...

Enforcers e.g. `Sort`, `Partition`.

Terminology (2/2)

Rule = (`CheckPattern`, `Transform`):

- Transformation rule, e.g. $R \bowtie S \quad \rightarrow \quad S \bowtie R$
- Implementation rule, e.g. $R \bowtie S \quad \rightarrow \quad R$ `HashJoin` S

Memo = E-Graph

Group or Class = E-Class

Expression = E-Node

Volcano

1. Generation Phase: repeatedly applies transformation rules until saturation.

2. Cost Analysis Phase: applies implementation rules, using a cost `Limit`

R(A,B), S(B,C), T(C,D)

The Memo

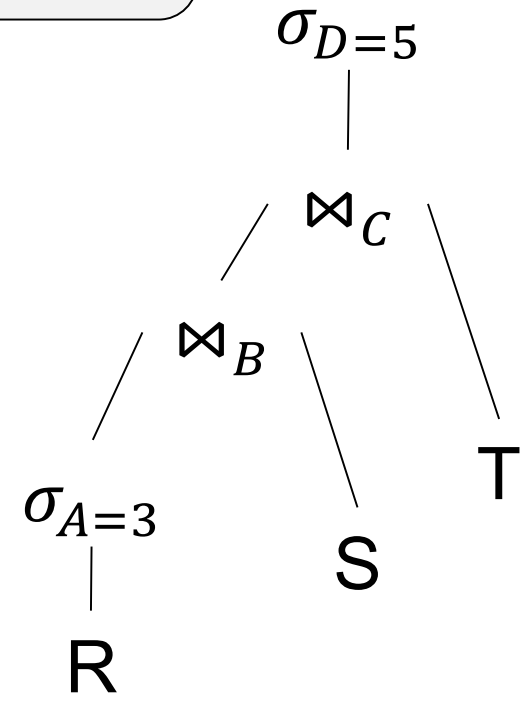
```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

Initialize Memo w/ one (naïve) plan



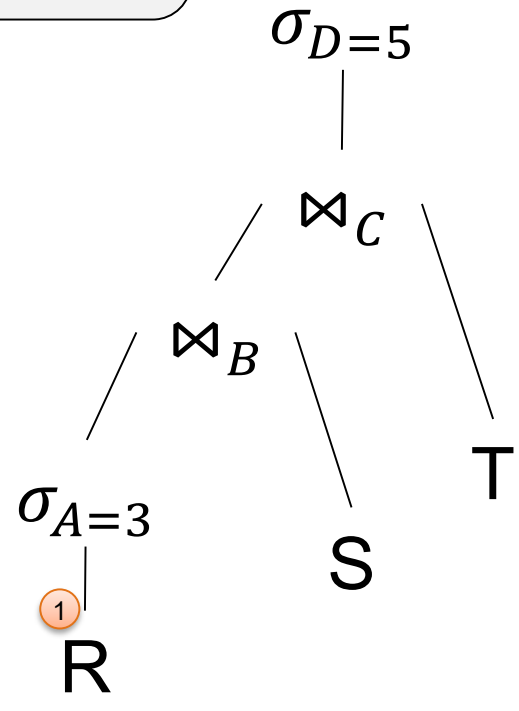
R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1 Scan R

Initialize Memo w/ one (naïve) plan



R(A,B), S(B,C), T(C,D)

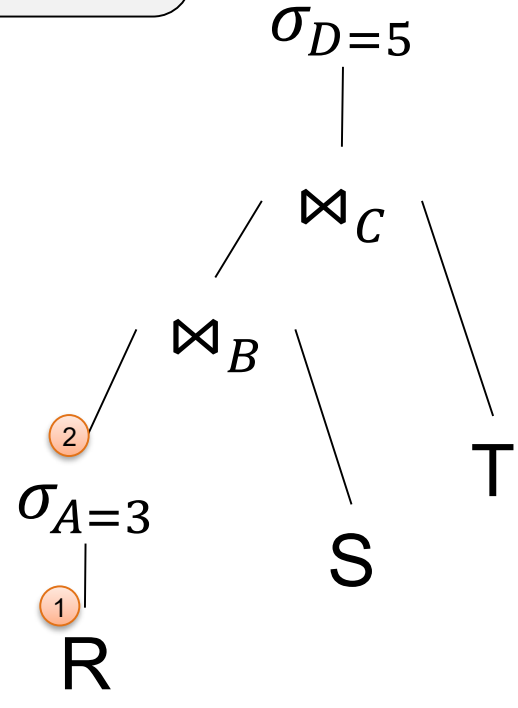
```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1 Scan R

2 Select[A=3] 1

Initialize Memo w/ one (naïve) plan



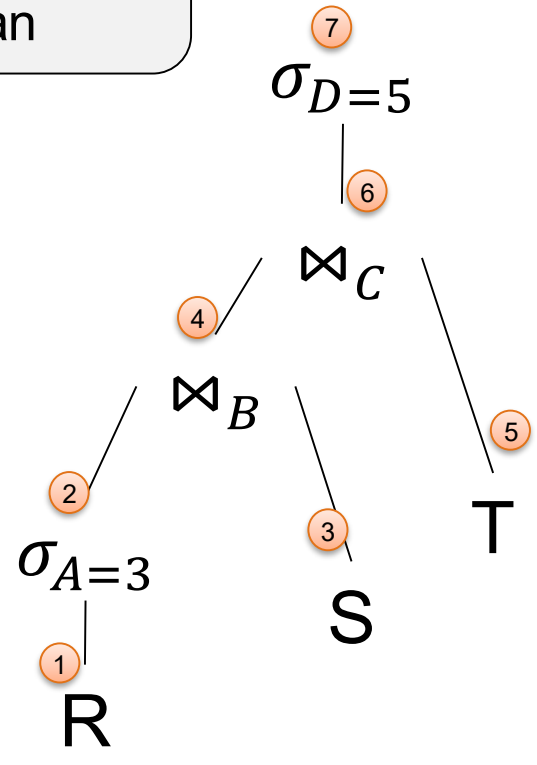
R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R
2	Select[A=3] 1
3	Scan S
4	Join[B=B] 2, 3
5	Scan T
6	Join[C=C] 4,5
7	Select[D=5] 6

Initialize Memo w/ one (naïve) plan

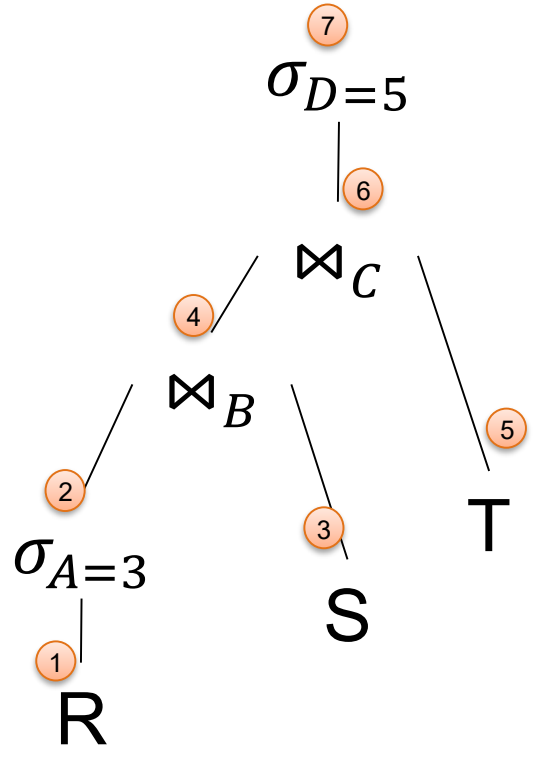


R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R
2	Select[A=3] 1
3	Scan S
4	Join[B=B] 2, 3
5	Scan T
6	Join[C=C] 4,5
7	Select[D=5] 6



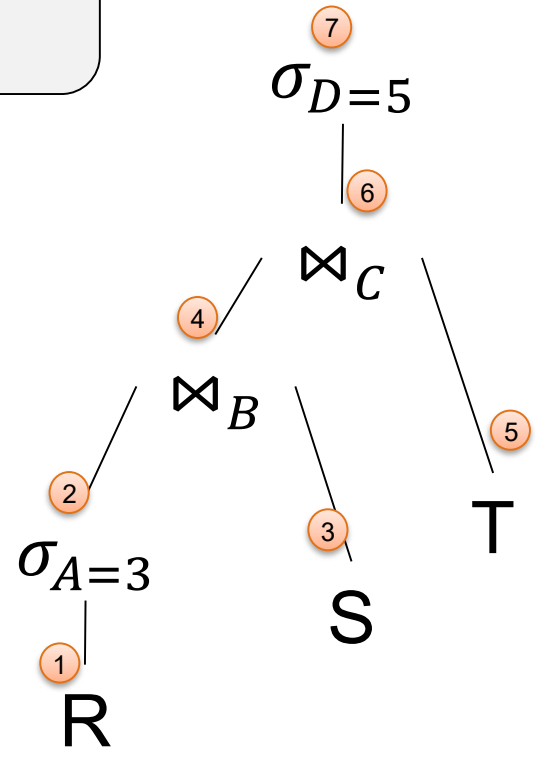
R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R
2	Select[A=3] 1
3	Scan S
4	Join[B=B] 2, 3
5	Scan T
6	Join[C=C] 4,5
7	Select[D=5] 6

Apply an optimization rule



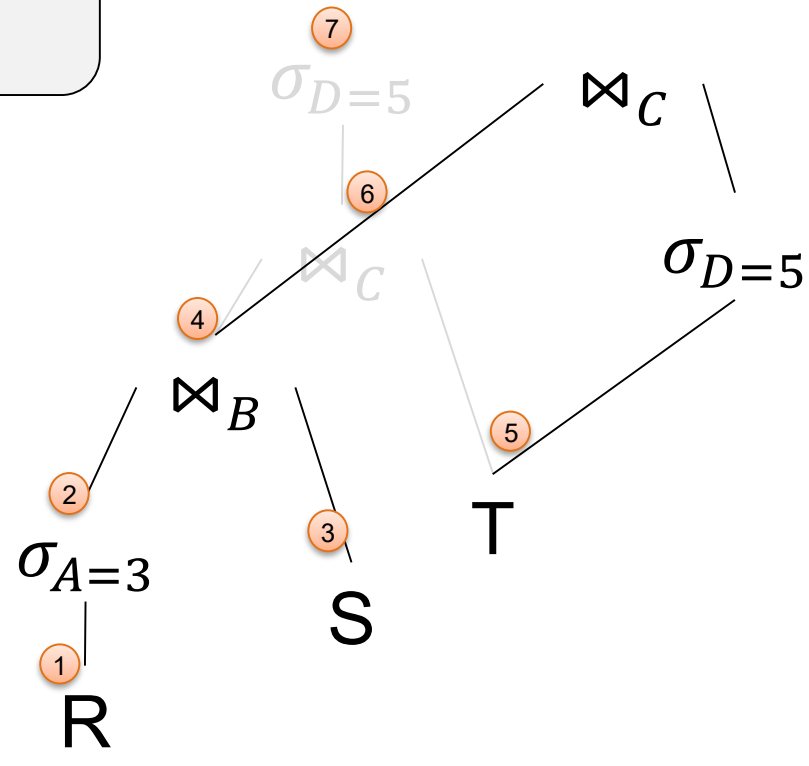
R(A,B), S(B,C), T(C,D)

The Memo

1	Scan R
2	Select[A=3] 1
3	Scan S
4	Join[B=B] 2, 3
5	Scan T
6	Join[C=C] 4,5
7	Select[D=5] 6

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

Apply an optimization rule

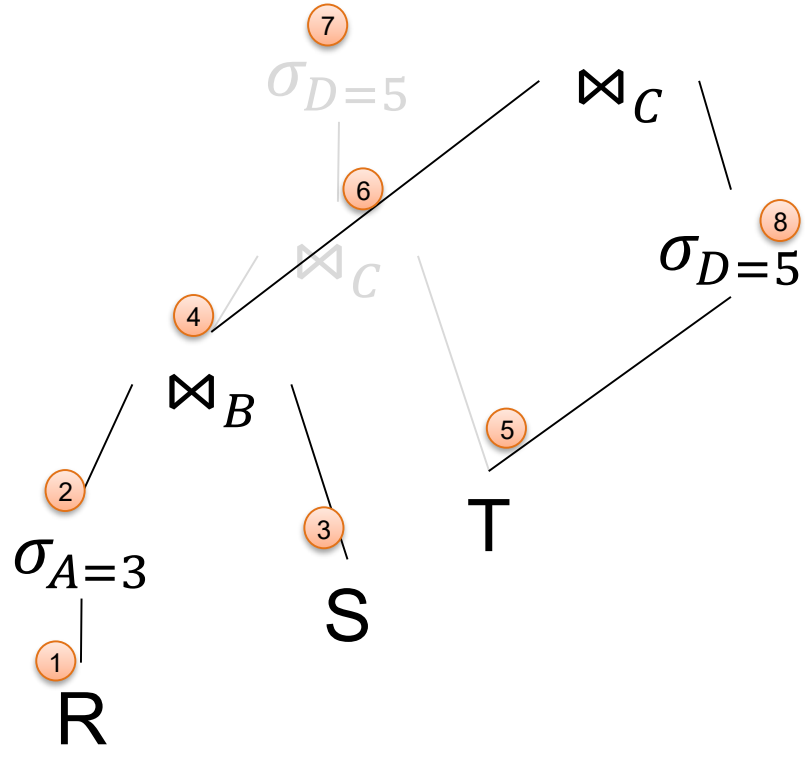


R(A,B), S(B,C), T(C,D)

The Memo

1	Scan R
2	Select[A=3] 1
3	Scan S
4	Join[B=B] 2, 3
5	Scan T
6	Join[C=C] 4,5
7	Select[D=5] 6
8	Select[D=5] 5

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

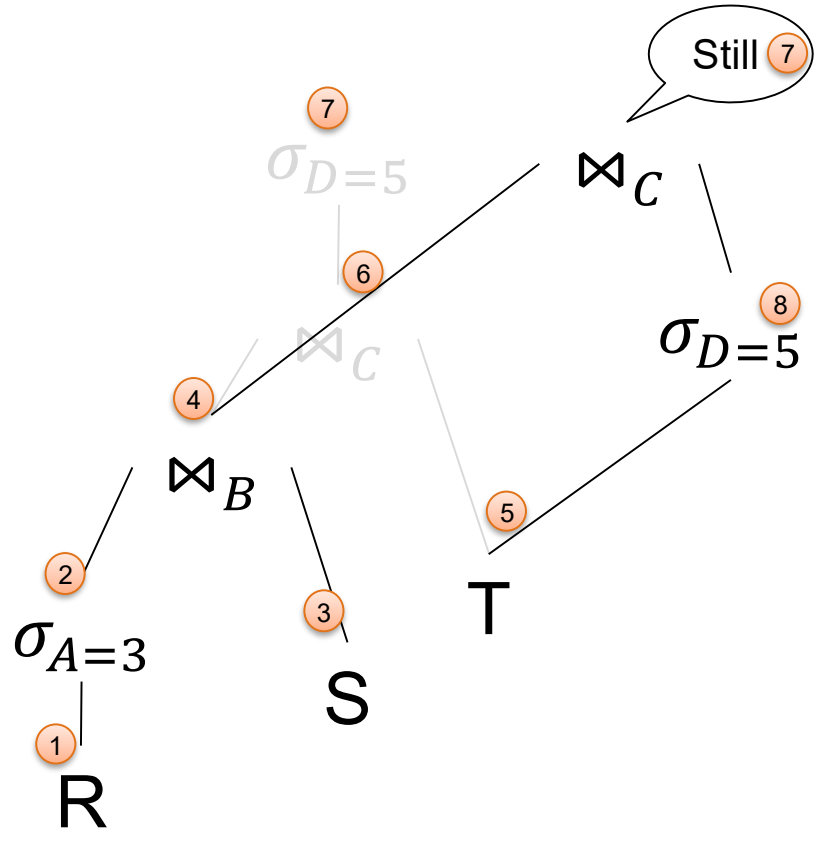


R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R	
2	Select[A=3] 1	
3	Scan S	
4	Join[B=B] 2, 3	
5	Scan T	
6	Join[C=C] 4,5	
7	Select[D=5] 6	Join[C=C] 4,8
8	Select[D=5] 5	

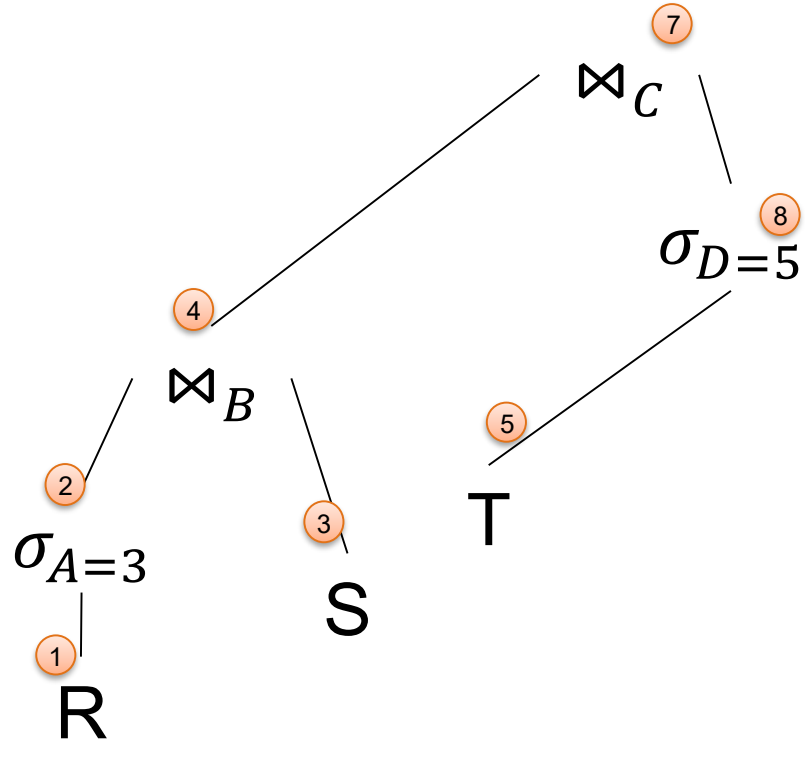


R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R	
2	Select[A=3] 1	
3	Scan S	
4	Join[B=B] 2, 3	
5	Scan T	
6	Join[C=C] 4,5	
7	Select[D=5] 6	Join[C=C] 4,8
8	Select[D=5] 5	

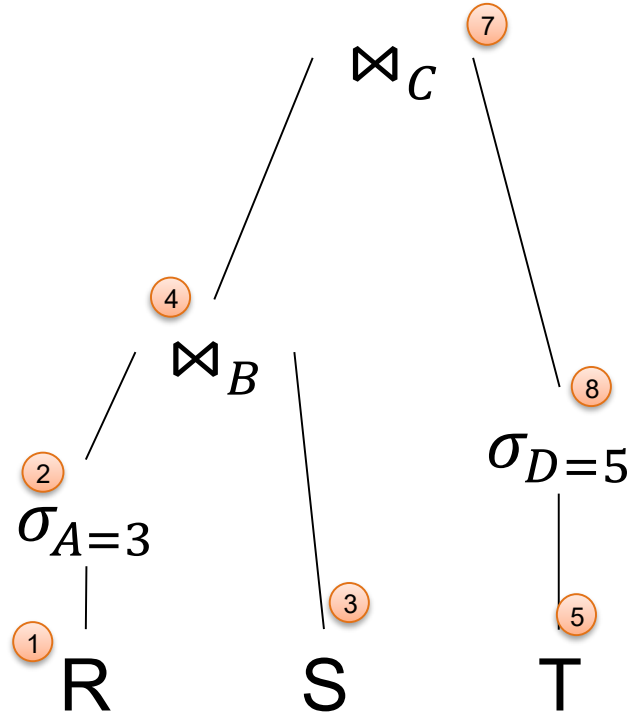


R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R	
2	Select[A=3] 1	
3	Scan S	
4	Join[B=B] 2, 3	
5	Scan T	
6	Join[C=C] 4,5	
7	Select[D=5] 6	Join[C=C] 4,8
8	Select[D=5] 5	



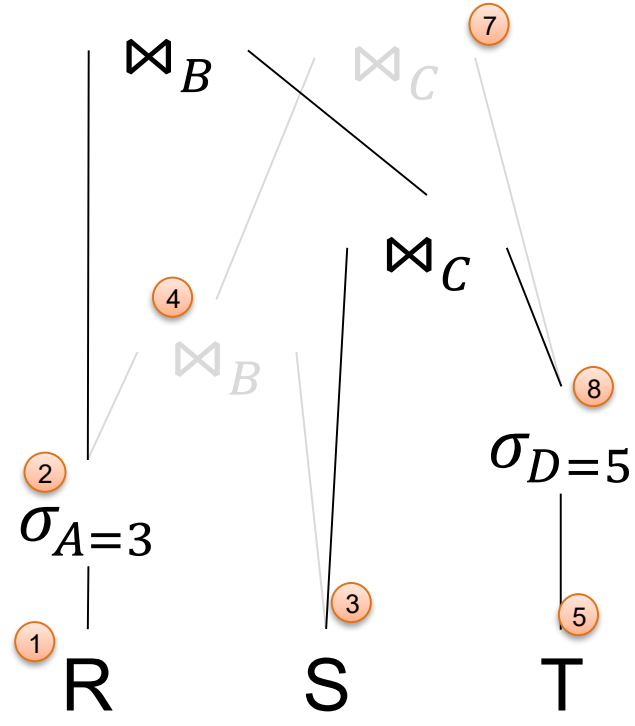
R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R	
2	Select[A=3] 1	
3	Scan S	
4	Join[B=B] 2, 3	
5	Scan T	
6	Join[C=C] 4,5	
7	Select[D=5] 6	Join[C=C] 4,8
8	Select[D=5] 5	

Apply another rule

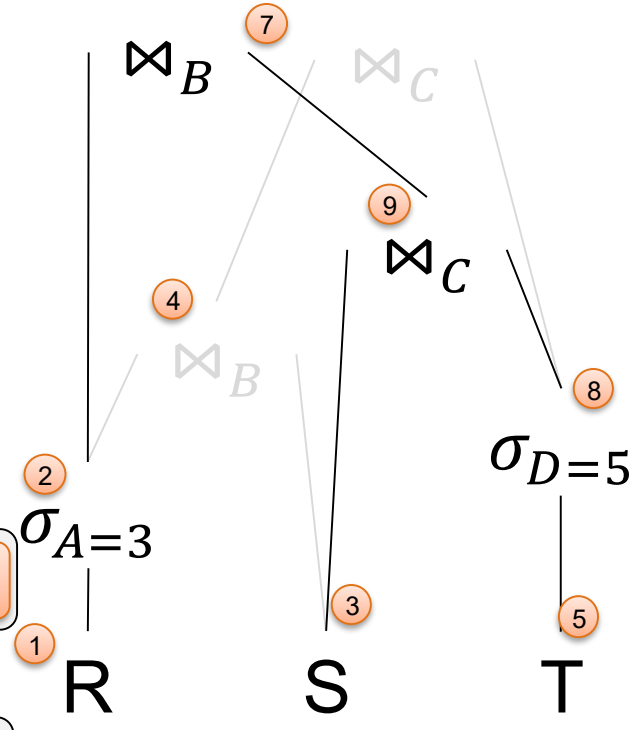


R(A,B), S(B,C), T(C,D)

```
select *
from R, S, T
where R.B=S.B
    and S.C=T.C
    and R.A = 3
    and T.D = 5
```

The Memo

1	Scan R		
2	Select[A=3] 1		
3	Scan S		
4	Join[B=B] 2, 3		
5	Scan T		
6	Join[C=C] 4,5		
7	Select[D=5] 6	Join[C=C] 4,8	Join[B=B] 2,9
8	Select[D=5] 5		
9	Join[C=C] 3, 8		



Volcano: Generation Phase

[Ding et al., 2024] Top Down equality saturation:

```
1: function GenerateLogicalExpr(LogExpr, Rules)        ▷ Generation phase
2:     for Child in inputs of LogExpr do
3:         if Group(Child) ∉ Memo then
4:             GenerateLogicalExpr(Child)                    ▷ Update memo
5:     MatchTransRule(LogExpr, Rules)
6: function MatchTransRule(LogExpr, Rules)   ▷ Apply transformation rules
7:     for rule in Rules do
8:         if rule matches LogExpr then
9:             NewLogExpr ← Transform(LogExpr, rule)   ▷ Update memo and
   record the neighbors
10:            GenerateLogicalExpr(NewLogExpr)          ▷ Only invoke if
   NewLogExpr is not previously in memo
```

Small bug: `GenerateLogicalExpr` should have `Rules` as parameter.

Volcano: Cost Analysis Phase

`FindBestPlan`: Logical Expression $\mapsto$ Physical plan

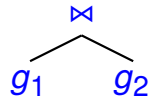
`FindBestPlan`(`Group`, `Limit`) by example:

Volcano: Cost Analysis Phase

`FindBestPlan`: Logical Expression $\mapsto$ Physical plan

`FindBestPlan(Group, Limit)` by example:

- For each Expr $\in$ `Group`

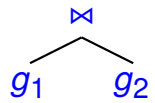


Volcano: Cost Analysis Phase

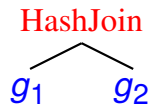
`FindBestPlan`: Logical Expression $\mapsto$ Physical plan

`FindBestPlan(Group, Limit)` by example:

- For each Expr $\in$ `Group`



- Apply an implementation rule



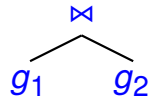
Cost = 100

Volcano: Cost Analysis Phase

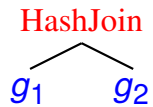
`FindBestPlan`: Logical Expression $\mapsto$ Physical plan

`FindBestPlan(Group, Limit)` by example:

- For each Expr $\in$ `Group`

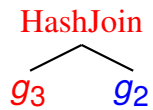


- Apply an implementation rule



Cost = 100

- `FindBestPlan`(g_1, `Limit` $-$ 100)



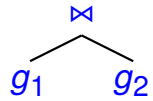
Cost = 2100

Volcano: Cost Analysis Phase

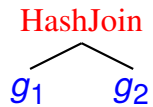
`FindBestPlan`: Logical Expression $\mapsto$ Physical plan

`FindBestPlan(Group, Limit)` by example:

- For each Expr $\in$ `Group`

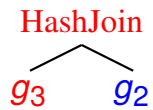


- Apply an implementation rule



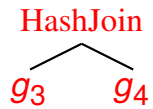
Cost = 100

- `FindBestPlan`(g_1, `Limit` $-$ 100)



Cost = 2100

- `FindBestPlan`(g_2, `Limit` $-$ 2100)



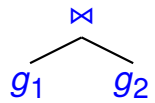
Cost = . . .

Volcano: Cost Analysis Phase

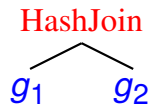
`FindBestPlan`: Logical Expression $\mapsto$ Physical plan

`FindBestPlan(Group, Limit)` by example:

- For each Expr $\in$ `Group`

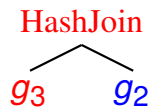


- Apply an implementation rule



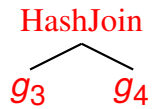
Cost = 100

- `FindBestPlan`(g_1, `Limit` $-$ 100)



Cost = 2100

- `FindBestPlan`(g_2, `Limit` $-$ 2100)



Cost = . . .

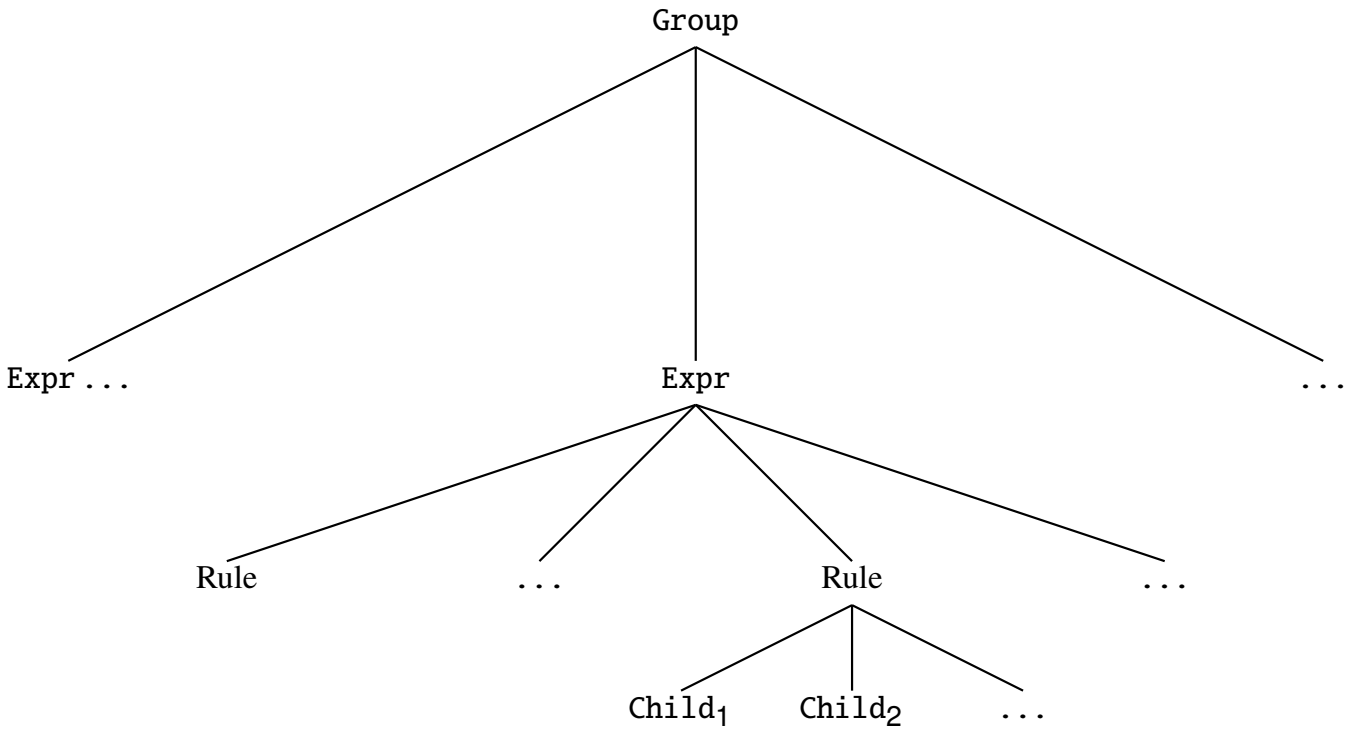
- Whenever Cost > Limit stop and return NULL

Volcano: Cost Analysis Phase (Very Simplified)

Function FindBestPlan(Group, Limit)

```
if Limit ≤ 0 then return Null;
Plan, Cost := LookUpMemo(Group)
if Cost ≠ Null then
    if Cost ≤ Limit then return Plan,Cost ;
    else return NULL;
Cost := ∞;
for Expr ∈ Group, r ∈ ImplementationRules(Expr) do
    Pln, Cst := r.Op, r.Cost ;          // Start new plan with r.Op
    for Grp ∈ Children(Expr) do
        Pln, Cst += FindBestPlan(Grp, Limit − Cst)
    if Success all children? then Plan, Cost, Limit := Pln, Cst, Cst;;
InsertMemo(Group, Plan, Cost)
return (Plan, Cost)
```

Volcano: Cost Analysis Phase (Visual Representation)



$$\texttt{Limit}_1 \geq \texttt{Limit}_2 \geq \texttt{Limit}_3 \geq \cdots$$

When `Cost` > `Limit`, prune the search.

Discussion

- Generation phase is straightforward: saturate the whole E-Graph.
- Cost-analysis phase: prunes the search based on cost. In addition:
  - Store applicable implementation rules in a set of `Moves`
  - Sort `Moves` by "promise", to find better plans first.
  - Early stopping is possible, e.g. timeout.
- A physical expression may have multiple version, with different properties.

Cascades

Will discuss only at a very high level.

Cascades makes the following changes to Volcano:

- Handles transformation and implementation rules in the same phase.
- Replaces recursive call with stack-based processing: allows control over the order of operations.
- Breaks down each rule application into smaller steps, each becomes a separate task, to be placed on the stack.

This allows for a more aggressive cost-based pruning.

Discussion

- Volcano/Cascades: complex, but clean foundation in E-Graphs
- Lots of additional heuristics:
  - Simplification rules: apply $E_1 \rightarrow E_2$, then delete E_1
  - Macro rules: e.g. $\texttt{StarJoin}(R_1, \ldots, R_n) \rightarrow \texttt{DPccp}$
- Systems implementing Cascades: SQLServer, Orca (Greenplum), Calcite.
- Other systems: Postgres (DP), Catalyst/Spark (greedy).


Baader, F. and Nipkow, T. (1999).
Term Rewriting and All That.
Cambridge University Press, USA.


Ding, B., Narasayya, V. R., and Chaudhuri, S. (2024).
Extensible query optimizers in practice.
Found. Trends Databases, 14(3-4):186–402.


Graefe, G. (1994).
Volcano - an extensible and parallel query evaluation system.
IEEE Trans. Knowl. Data Eng., 6(1):120–135.


Graefe, G. (1995).
The cascades framework for query optimization.
IEEE Data Eng. Bull., 18(3):19–29.


McKenna, W. J. (1993).
Efficient Search in Extensible Query Optimization: The Volcano Optimizer Generator.
PhD thesis, University of Colorado-Boulder.


Willsey, M., Nandi, C., Wang, Y. R., Flatt, O., Tatlock, Z., and Panchekha, P. (2021).
egg: Fast and extensible equality saturation.
Proc. ACM Program. Lang., 5(POPL):1–29.


Xu, Y. (1998).
Efficiency in the Columbia Database Query Optimizer.
PhD thesis, Portland State University.