# CSE599d: Advanced Query Processing

# Lecture 9: Extensible Query Optimizers

Dan Suciu

University of Washington

# Top-Down v.s. Bottom-Up

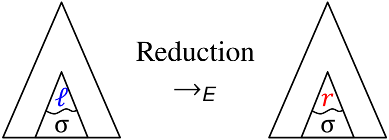
- Volcano/Cascades perform optimization Top-Down.
- Memoization can blur the distinction between top-down and bottom-up. E.g. recall top-down v.s. bottom-up Dynamic Programming in Lecture 3:
  - Bottom up: iterate over subsets $S \subseteq \{R_1, \ldots, R_n\}$, from smaller to larger.
  - Top down: partition a set into $S = S_1 \cup S_2$, process recursively $S_1$ and $S_2$, memoize all results.

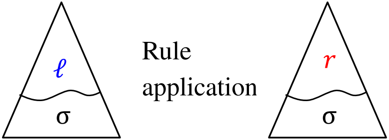
What are the pros and cons of Top-down v.s. Bottom-up?

# Reduction v.s. Rule Application

Top-down evaluation applies rules only at the top:

$$\ell \approx r$$




$$C[\sigma(\ell)] \rightarrow_E C[\sigma(r)] \qquad \sigma(\ell) \rightarrow \sigma(r)$$

Top-down optimizers use only rule application

# Review: Volcano's Generation Phase

```
Function GenerateLogicalExpr(LogicalExpr)
if LogicalExpr ∈ Memo then return Group(LogicalExpr);
for Child in children(LogicalExpr) do GenerateLogicalExpr(Child) ;
Insert(Group(LogicalExpr), Memo);
for r ∈ Rules do
    if Matches(r, LogicalExpr) then
        NewLogicalExpr := Apply(r, LogicalExpr);
        GenerateLogicalExpr(NewLogicalExpr);
```

# Volcano: Cost Analysis Phase

FindBestPlan: Logical Expression ↦ Physical plan

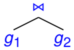
FindBestPlan(Group, Limit) by example:

# Volcano: Cost Analysis Phase

`FindBestPlan`: Logical Expression ↦ Physical plan

`FindBestPlan(Group, Limit)` by example:

- For each Expr ∈ `Group`

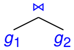


# Volcano: Cost Analysis Phase

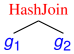
`FindBestPlan`: Logical Expression $\mapsto$ Physical plan

`FindBestPlan(Group, Limit)` by example:

- For each Expr $\in$ `Group`




- Apply an implementation rule



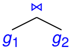

Cost = 100

# Volcano: Cost Analysis Phase

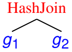
`FindBestPlan`: Logical Expression $\mapsto$ Physical plan

`FindBestPlan(Group, Limit)` by example:

- For each Expr $\in$ `Group`

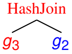



- Apply an implementation rule




Cost = 100

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



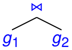

Cost = 2100

# Volcano: Cost Analysis Phase

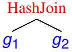
FindBestPlan: Logical Expression ↦ Physical plan

FindBestPlan(Group, Limit) by example:

- For each Expr ∈ Group

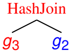



- Apply an implementation rule



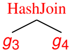

Cost = 100

- FindBestPlan($g_1$, Limit − 100)




Cost = 2100

- FindBestPlan($g_2$, Limit − 2100)



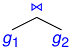

Cost = . . .

# Volcano: Cost Analysis Phase

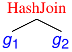
`FindBestPlan`: Logical Expression ↦ Physical plan

`FindBestPlan(Group, Limit)` by example:

- For each Expr ∈ `Group`

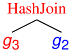



- Apply an implementation rule



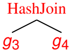

Cost = 100

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




Cost = 2100

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




Cost = ...

- Whenever Cost > Limit stop and return NULL

# Discussion

- Volcano: separate logical and physical phase; both are recursive, top-down. Early pruning in the physical phase.
- Cascades: combine logical/physical phase, stack-based; more aggressive early pruning.
- LOTS of engineering/heuristics: simplification rules, macro rules, prioritize by “promise”.
- Wide adoption: SQLServer, Orca (Greenplum), Calcite, Cockroach Lab.

# Complexity of Rule-Based Transformation

# Overview

Join reordering: $\Sigma = \{\bowtie\}$
What is the complexity of a Transformation Based v.s. Dynamic Programming?

Two scenarios:

- Bushy plans with cross products[1] [Pellenkoft et al., 1997]
- Bushy plans without cross products [Shanbhag and Sudarshan, 2014]

[1] Equivalently: assume the query is fully connected.

# Review: Space/Time Complexity of DP

$n$ input relations $R_1, R_2, \ldots, R_n$

- Space complexity (store all subsets $S$): ?????
- Time complexity (enumerate all disjoint subsets $S_1, S_2$): ?????

# Review: Space/Time Complexity of DP

$n$ input relations $R_1, R_2, \ldots, R_n$

- Space complexity (store all subsets $S$): $2^n$

- Time complexity (enumerate all disjoint subsets $S_1, S_2$): ?????

# Review: Space/Time Complexity of DP

$n$ input relations $R_1, R_2, \ldots, R_n$

- Space complexity (store all subsets $S$): $2^n$
- Time complexity (enumerate all disjoint subsets $S_1, S_2$): $3^n$

# Review: Space/Time Complexity of DP

$n$ input relations $R_1, R_2, \ldots, R_n$

- Space complexity (store all subsets $S$): $2^n$
  More precisely: $2^n - 1$

- Time complexity (enumerate all disjoint subsets $S_1, S_2$): $3^n$
  More precisely: $3^n - 2 \cdot 2^n + 1$

# Review: Space/Time Complexity of DP

$n$ input relations $R_1, R_2, \ldots, R_n$

- Space complexity (store all subsets $S$): $2^n$
  More precisely: $2^n - 1$
- Time complexity (enumerate all disjoint subsets $S_1, S_2$): $3^n$
  More precisely: $3^n - 2 \cdot 2^n + 1$
- DPccp: reduction to $O(n^2)$ and $O(n^3)$ when the query graph is a chain.

# Complexity of Associativity and Commutativity

**RS-B0** (Rule Set for Bushy trees)

Right Associativity:
$(A \bowtie B) \bowtie C \to A \bowtie (B \bowtie C)$

Left Associativity:
$A \bowtie (B \bowtie C) \to (A \bowtie B) \bowtie C$

Commutativity:
$A \bowtie B \to B \bowtie A.$

# Problem Statement

Find the space/ time complexity for RS-B0 on $R_1, \ldots, R_n$. Two scenarios:

- Explore all bushy plans.
- Explore bushy plans w/o cross products.

# All Bushy Plans

# Size of the Memo

$n$ tables:

- $2^n$ E-classes
- $3^n$ E-nodes

$$
\begin{aligned}
\text{abcd} &= [a] \bowtie [bcd]; [b] \bowtie [acd]; [c] \bowtie [abd];\\
&\quad [d] \bowtie [abc]; [ab] \bowtie [cd]; [ac] \bowtie [bd];\\
&\quad [ad] \bowtie [bc]; [bcd] \bowtie [a]; [acd] \bowtie [b];\\
&\quad [abd] \bowtie [c]; [abc] \bowtie [d]; [cd] \bowtie [ab];\\
&\quad [bd] \bowtie [ac]; [bc] \bowtie [ad].\\
\text{abc} &= [a] \bowtie [bc]; [b] \bowtie [ac]; [c] \bowtie [ab];\\
&\quad [bc] \bowtie [a]; [ac] \bowtie [b]; [ab] \bowtie [c].\\
\text{abd} &= [a] \bowtie [bd]; [b] \bowtie [ad]; [d] \bowtie [ab];\\
&\quad [bd] \bowtie [a]; [ad] \bowtie [b]; [ab] \bowtie [d].\\
\text{acd} &= [a] \bowtie [cd]; [c] \bowtie [ad]; [d] \bowtie [ac];\\
\text{acd} &= [cd] \bowtie [a]; [ad] \bowtie [c]; [ac] \bowtie [d].\\
\text{bcd} &= [b] \bowtie [cd]; [c] \bowtie [bd]; [d] \bowtie [bc];\\
&\quad [cd] \bowtie [b]; [bd] \bowtie [c]; [bc] \bowtie [d].\\
\text{ab} &= [a] \bowtie [b]; [b] \bowtie [a].\\
\text{ac} &= [a] \bowtie [c]; [c] \bowtie [a].\\
\text{ad} &= [a] \bowtie [d]; [d] \bowtie [a].\\
\text{bc} &= [b] \bowtie [c]; [c] \bowtie [b].\\
\text{bd} &= [b] \bowtie [d]; [d] \bowtie [b].\\
\text{cd} &= [c] \bowtie [d]; [d] \bowtie [c].
\end{aligned}
$$

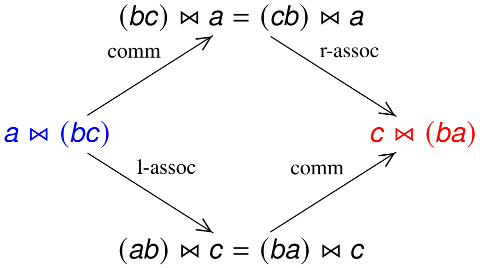
# Space Complexity

The space complexity of a transformation-based system is $O(3^n)$

The time complexity may be higher, because of duplicated work.

# Duplicated Work

Trivial duplicated work due to having two associativity rules:



# A Better Rule-set

**RS-B1** Keep only one associativity

Left Associativity:
$A \bowtie (B \bowtie C) \rightarrow (A \bowtie B) \bowtie C$

Commutativity:
$A \bowtie B \rightarrow B \bowtie A.$

# The Search Graph

The Search Graph:

- Its nodes are the E-nodes of $G$.
- The edges are pairs of E-nodes $(n_1, n_2)$ such that there is a transformation mapping $n_1$ to $n_2$.

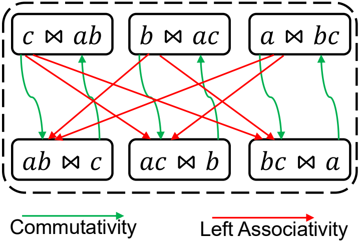
The edges of the search graph represent all steps of the algorithm.

# Example

Fragment of the Search Graph consisting of the E-nodes in the E-class $\boxed{abc}$

6 nodes

12 edges




The children of $\boxed{c \bowtie ab}$ are E-classes $\boxed{c}$ and $\boxed{ab}$

The E-class $\boxed{ab}$ contains $\boxed{a \bowtie b}$ and $\boxed{b \bowtie a}$

# Time Complexity of RS-B1

Count # o edges in E-Class($S$), where $S \subseteq \{R_1, \ldots, R_n\}$ has size $k$.

Commutativity $S_1 \bowtie S_2 \rightarrow S_2 \bowtie S_1$, for partitions $S = S_1 \cup S_2$.

$$\boxed{2^k - 2}$$

Left assoc $S_1 \bowtie (S_2 S_3) \rightarrow (S_1 S_2) \bowtie S_3$, for partitions $S = S_1 \cup S_2 \cup S_3$,

$$\boxed{3^k - 2 \cdot 2^k + 1}$$

# Time Complexity of RS-B1

Count # o edges in E-Class($S$), where $S \subseteq \{R_1, \ldots, R_n\}$ has size $k$.

Commutativity $S_1 \bowtie S_2 \rightarrow S_2 \bowtie S_1$, for partitions $S = S_1 \cup S_2$.

$$\boxed{2^k - 2}$$

Left assoc $S_1 \bowtie (S_2 S_3) \rightarrow (S_1 S_2) \bowtie S_3$, for partitions $S = S_1 \cup S_2 \cup S_3$,

$$\boxed{3^k - 2 \cdot 2^k + 1}$$

Total edges = $\sum_{k=0,n} \binom{n}{k} (\cdots) = O(4^n)$

Time complexity: $O(4^n)$

Need a better Rule Set!

# A Better Rule Set (1/2)

RS-B2[2]

**Rule set RS-B2**:

- $R_1$ (Commutativity): $A \bowtie_0 B \to B \bowtie_1 A$
  Disable application of $R_1, R_2, R_3, R_4$ on new operator $\bowtie_1$
- $R_2$ (Left Associativity):
  $A \bowtie_0 (B \bowtie_1 C) \to (A \bowtie_2 B) \bowtie_3 C$
  Disable application of $R_2, R_3, R_4$ on new operator $\bowtie_3$
- $R_3$ (Right Associativity):
  $(A \bowtie_0 B) \bowtie_1 C \to A \bowtie_2 (B \bowtie_3 C)$
  Disable application of $R_2, R_3, R_4$ on new operator $\bowtie_2$
- $R_4$ (Exchange):
  $(A \bowtie_0 B) \bowtie_1 (C \bowtie_2 D) \to (A \bowtie_3 C) \bowtie_4 (B \bowtie_5 D)$
  Disable application of $R_1, R_2, R_3, R_4$ on new operator $\bowtie_4$

[2] From [Shanbhag and Sudarshan, 2014]; identical to the original [Pellenkoft et al., 1997]

# A Better Rule Set (2/2)

"Disable application . . ." means only for current E-class.

RS-B2 revisited:

$R_1$ : Commutativity $A \bowtie B \rightarrow B \bowtie A$

$R_2$ : Left Associativity $A \bowtie (BC) \rightarrow (AB) \bowtie C$

$R_3$ : Right Associativity $(AB) \bowtie C \rightarrow A \bowtie (BC)$

$R_4$ : Exchange $(AB) \bowtie (CD) \rightarrow (AC) \bowtie (BD)$

$R_1, R_2, R_3, R_4$ : don't apply to $\bowtie$. $R_2, R_3, R_4$ : don't apply to $\bowtie$

# Search Graph Revisited

Algorithm starts from an initial expression, then applies transformations.

Example: starting from E-node $c \bowtie ab$, we generate the E-class $abc$ as follows:

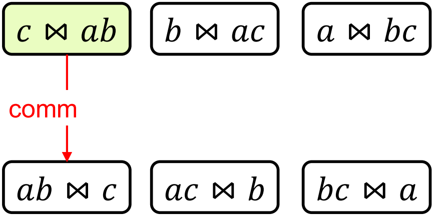
$c \bowtie ab$ $b \bowtie ac$ $a \bowtie bc$

$ab \bowtie c$ $ac \bowtie b$ $bc \bowtie a$

# Search Graph Revisited

Algorithm starts from an initial expression, then applies transformations.

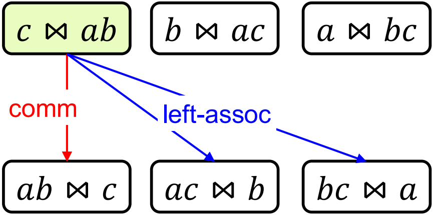
Example: starting from E-node $c \bowtie ab$, we generate the E-class $abc$ as follows:



# Search Graph Revisited

Algorithm starts from an initial expression, then applies transformations.

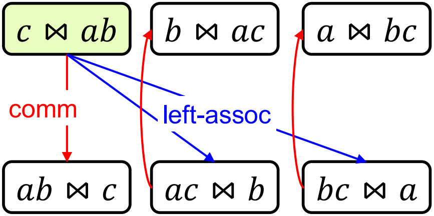
Example: starting from E-node $c \bowtie ab$, we generate the E-class *abc* as follows:



# Search Graph Revisited

Algorithm starts from an initial expression, then applies transformations.

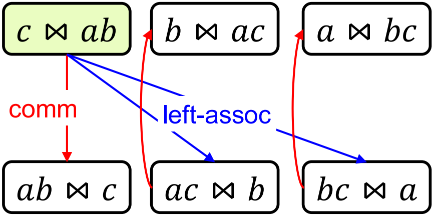
Example: starting from E-node $c \bowtie ab$, we generate the E-class *abc* as follows:



# Search Graph Revisited

Algorithm starts from an initial expression, then applies transformations.

Example: starting from E-node $c \bowtie ab$, we generate the E-class $abc$ as follows:




We didn't need Exchange! $(AB) \bowtie (CD) \rightarrow (AC) \bowtie (BD)$ Redundant????

# Completeness of RS-B2

**Theorem** The rules RS-B2 are complete for generating all bushy plans

# Completeness of RS-B2

**Theorem** The rules RS-B2 are complete for generating all bushy plans

**Proof** For any two partitions

$$S = S_1 \cup S_2 = S_3 \cup S_4$$

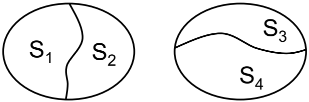
there exists a rewriting

$$S_1 \bowtie S_2 \xrightarrow{*}_{\text{RS-B2}} S_3 \bowtie S_4$$

# Completeness of RS-B2

**Theorem** The rules RS-B2 are complete for generating all bushy plans

**Proof** For any two partitions




$$S = S_1 \cup S_2 = S_3 \cup S_4$$

there exists a rewriting

$$S_1 \bowtie S_2 \xrightarrow{*}_{\text{RS-B2}} S_3 \bowtie S_4$$

# Completeness of RS-B2

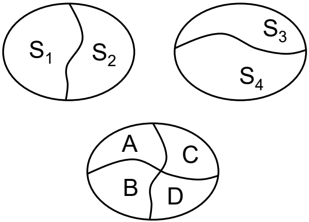
**Theorem** The rules RS-B2 are complete for generating all bushy plans

**Proof** For any two partitions

$$S = S_1 \cup S_2 = S_3 \cup S_4$$

there exists a rewriting

$$S_1 \bowtie S_2 \xrightarrow{*}_{\text{RS-B2}} S_3 \bowtie S_4$$



# Completeness of RS-B2

**Theorem** The rules RS-B2 are complete for generating all bushy plans

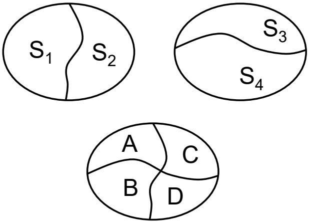
**Proof** For any two partitions

$$S = S_1 \cup S_2 = S_3 \cup S_4$$

there exists a rewriting

$$S_1 \bowtie S_2 \xrightarrow{*}_{\text{RS-B2}} S_3 \bowtie S_4$$

$$(AB) \bowtie (CD) \xrightarrow{*}_{\text{RS-B2}} (AC) \bowtie (BD)$$

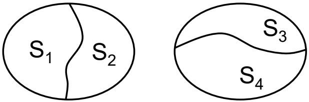


# Completeness of RS-B2

**Theorem** The rules RS-B2 are complete for generating all bushy plans

**Proof** For any two partitions

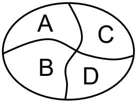
$$S = S_1 \cup S_2 = S_3 \cup S_4$$




there exists a rewriting

$$S_1 \bowtie S_2 \xrightarrow{*}_{\text{RS-B2}} S_3 \bowtie S_4$$

$$(AB) \bowtie (CD) \xrightarrow{*}_{\text{RS-B2}} (AC) \bowtie (BD)$$




**Case 1:** If $A, B, C, D$ are non-empty, then use $R_4$ Exchange:

$$(AB) \bowtie (CD) \rightarrow (AC) \bowtie (BD)$$

# Completeness of RS-B2

**Theorem** The rules RS-B2 are complete for generating all bushy plans

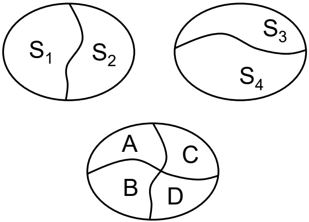
**Proof** For any two partitions

$$S = S_1 \cup S_2 = S_3 \cup S_4$$

there exists a rewriting

$$S_1 \bowtie S_2 \xrightarrow{*}_{\text{RS-B2}} S_3 \bowtie S_4$$

$$(AB) \bowtie (CD) \xrightarrow{*}_{\text{RS-B2}} (AC) \bowtie (BD)$$




**Case 1:** If $A, B, C, D$ are non-empty, then use $R_4$ Exchange:

$$(AB) \bowtie (CD) \rightarrow (AC) \bowtie (BD)$$

**Case 2:** One is empty, e.g. $D = \emptyset$. Use the previous example:

$$(AB) \bowtie C \rightarrow_{\text{Right-ass}} B \bowtie (AC) \rightarrow_{\text{Comm}} (AC) \bowtie B$$

# No Duplicate Work in RS-B2

**Theorem** RS-B2 generates each E-node exactly once

**Proof** Starting from $S_1 \bowtie S_2$ there are only two ways to reach $S_3 \bowtie S_4$:

$$S_1 \bowtie S_2 \rightarrow S_4 \bowtie S_3 \rightarrow S_3 \bowtie S_4$$
$$S_1 \bowtie S_2 \rightarrow S_3 \bowtie S_4$$

A simple case analysis (based on the sets $A, B, C, D$ from before) shows that exactly one of these two ways is possible for any given $S_3, S_4$.

# Time and Space Complexity

**Theorem** The time and space complexity of transformation-based search using RS-B2 is $O(3^n)$.

Proof: there are $3^n$ E-nodes, and every E-node (other than those in the original expression) has exactly one incoming edge in the Search Graph.

# No Cross Products

# Overview

Restricting expressions to plans without cross products may reduce the search space significantly. Recall DPsub v.s.t DPccp.

Are the rule sets RS-B0, RS-B1, RS-B2 still complete if we restrict all expressions to be without cross products?

# Notation

Fix a rule set RS.

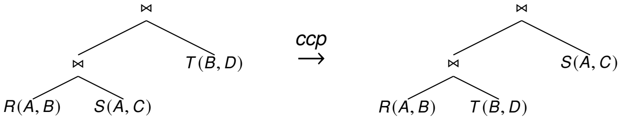
$\boxed{s \stackrel{ccp}{\rightarrow} t}$ means $s \stackrel{*}{\rightarrow} t$ and all intermediate steps are without cross product.

# Notation

Fix a rule set RS.

$\boxed{s \xrightarrow{ccp} t}$ means $s \xrightarrow{*} t$ and all intermediate steps are without cross product.

Prove (in class):

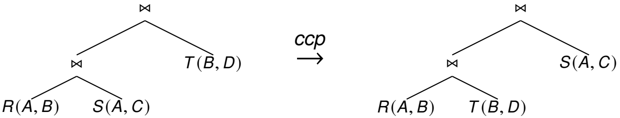


# Notation

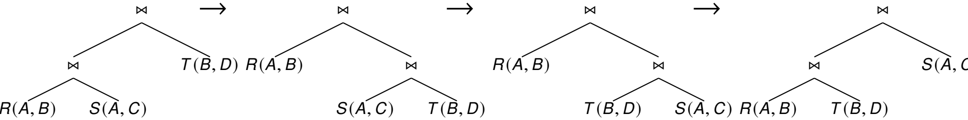
Fix a rule set RS.

$\boxed{s \overset{ccp}{\longrightarrow} t}$ means $s \overset{*}{\rightarrow} t$ and all intermediate steps are without cross product.

Prove (in class):




This does not work:



# Completeness of RS-B1 [Shanbhag and Sudarshan, 2014]

RS-B1: Commutativity and Left Associativity.

**Theorem** The rules RS-B1 are complete for bushy plans w/o cross product

# Completeness of RS-B1 [Shanbhag and Sudarshan, 2014]

RS-B1: Commutativity and Left Associativity.

**Theorem** The rules RS-B1 are complete for bushy plans w/o cross product

**Proof** Fix any left-linear join tree w/o cross product:

$$t_0 = ((R_1 \bowtie R_2) \bowtie R_3) \bowtie \cdots \bowtie R_n$$

# Completeness of RS-B1 [Shanbhag and Sudarshan, 2014]

RS-B1: Commutativity and Left Associativity.

**Theorem** The rules RS-B1 are complete for bushy plans w/o cross product

**Proof** Fix any left-linear join tree w/o cross product:

$$t_0 = ((R_1 \bowtie R_2) \bowtie R_3) \bowtie \cdots \bowtie R_n$$

Let $t$ be any bushy plan w/o cross products

**Lemma** There exists a reduction $t \overset{ccp}{\rightarrow} t_0$ w/o cross products.
Proof on next slide

# Completeness of RS-B1 [Shanbhag and Sudarshan, 2014]

RS-B1: Commutativity and Left Associativity.

**Theorem** The rules RS-B1 are complete for bushy plans w/o cross product

**Proof** Fix any left-linear join tree w/o cross product:

$$t_0 = ((R_1 \bowtie R_2) \bowtie R_3) \bowtie \cdots \bowtie R_n$$

Let $t$ be any bushy plan w/o cross products

**Lemma** There exists a reduction $t \stackrel{ccp}{\rightarrow} t_0$ w/o cross products.
Proof on next slide

**Corollary** There exists a reduction $t_0 \stackrel{ccp}{\rightarrow} t$ w/o cross products.
Prove in class (using the lemma)

# Proof of the Lemma

Let $t_0, t$ be w/o cross product, and $t_0$ is left-linear. Then $t \xrightarrow{ccp} t_0$

Proof by induction on $n$.

# Proof of the Lemma

Let $t_0, t$ be w/o cross product, and $t_0$ is left-linear. Then $t \stackrel{ccp}{\rightarrow} t_0$

Proof by induction on $n$.

Let $R_i$ be the sibling of $R_1$ in $t$: $t = \cdots (R_1 \bowtie R_i) \cdots$

# Proof of the Lemma

Let $t_0, t$ be w/o cross product, and $t_0$ is left-linear. Then $t \xrightarrow{ccp} t_0$

Proof by induction on $n$.

Let $R_i$ be the sibling of $R_1$ in $t$: $t = \cdots (R_1 \bowtie R_i) \cdots$

Replace $R_1 \bowtie R_i$ with new symbol $R_{1i}$: $t' = \cdots R_{1i} \cdots$

# Proof of the Lemma

Let $t_0, t$ be w/o cross product, and $t_0$ is left-linear. Then $t \stackrel{ccp}{\longrightarrow} t_0$

Proof by induction on $n$.

Let $R_i$ be the sibling of $R_1$ in $t$: $t = \cdots (R_1 \bowtie R_i) \cdots$

Replace $R_1 \bowtie R_i$ with new symbol $R_{1i}$: $t' = \cdots R_{1i} \cdots$

By induction: $t' \stackrel{ccp}{\longrightarrow} ((R_{1i} \bowtie R_2) \bowtie R_3) \bowtie \cdots R_{i-1} \bowtie R_{i+1} \bowtie \cdots \bowtie R_n$

# Proof of the Lemma

Let $t_0, t$ be w/o cross product, and $t_0$ is left-linear. Then $t \stackrel{ccp}{\longrightarrow} t_0$

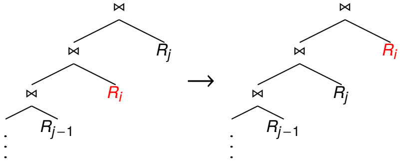
Proof by induction on $n$.

Let $R_i$ be the sibling of $R_1$ in $t$: $t = \cdots (R_1 \bowtie R_i) \cdots$

Replace $R_1 \bowtie R_i$ with new symbol $R_{1i}$: $t' = \cdots R_{1i} \cdots$

By induction: $t' \stackrel{ccp}{\longrightarrow} ((R_{1i} \bowtie R_2) \bowtie R_3) \bowtie \cdots R_{i-1} \bowtie R_{i+1} \bowtie \cdots \bowtie R_n$

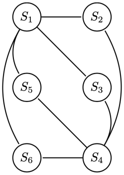
Move $R_i$ up step by step:



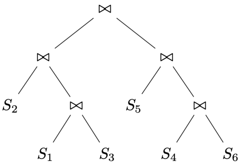

for all $j < i$

# Incompleteness of RS-B2 [Shanbhag and Sudarshan, 2014]

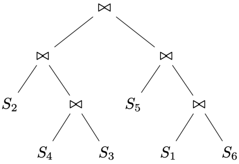
RS-B2: Rewrite system with restrictions $Q \xrightarrow{ccp} Q_2$ impossible.




(a) Join Graph J




(b) Query Tree $Q$




(c) Query Tree $Q_2$

Figure 5: Incompleteness of RS-B2-CPS

# Summary

# Summary

- Extensible Query Optimizers make it easy to add new operators.:
  - Solve the Word Problem using the E-graphs.
  - Lots of extensions and clever engineering.
- Clever rule design can decrease the complexity, but it likely remains higher than Dynamic Programming.
- Why is the Memo always acyclic? Why is Volcano guaranteed to terminate?

Pellenkoft, A., Galindo-Legaria, C. A., and Kersten, M. L. (1997).
The complexity of transformation-based join enumeration.
In Jarke, M., Carey, M. J., Dittrich, K. R., Lochovsky, F. H., Loucopoulos, P., and Jeusfeld, M. A., editors, VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece, pages 306–315. Morgan Kaufmann.

Shanbhag, A. and Sudarshan, S. (2014).
Optimizing join enumeration in transformation-based query optimizers.
Proc. VLDB Endow., 7(12):1243–1254.