CSE599d: Advanced Query Processing

Lecture 11: Yannakakis' Algorithm

Dan Suciu

University of Washington

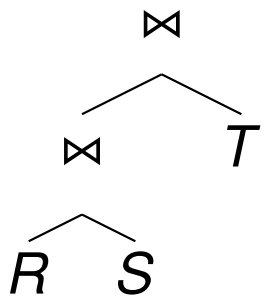
Announcements

- HW3 is posted. Due: Friday Feb. 27
- No lecture on Monday, Feb. 16: president's day.
- Today and next week: Yannakakis' algorithm.

Motivation

Diamond queries: the intermediate results grow, then shrink:

- Joins cause relations to grow.
- But later joins cause them to shrink.



Yannakakis Algorithm

Introduced by [Yannakakis, 1981]

YA compute an acyclic query in time $O(|IN| + |OUT|)$. Optimal!

Similar (almost identical) to the Junction Tree Algorithm in graphical models.

Acyclic Queries

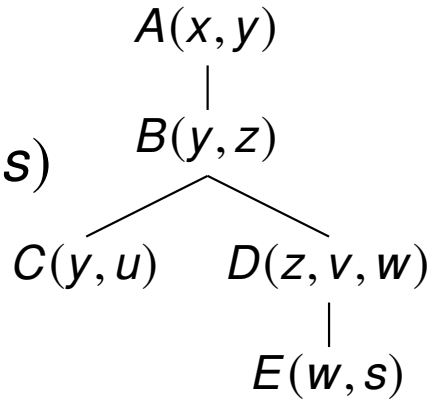
Acyclic Query

Q is acyclic if it admits a join tree, which is a tree T where:

- The nodes in T are in 1-1 correspondence with the atoms in Q.
- T satisfies the running intersection property: for any variable, the set of nodes that contain it forms a connected component.

Examples

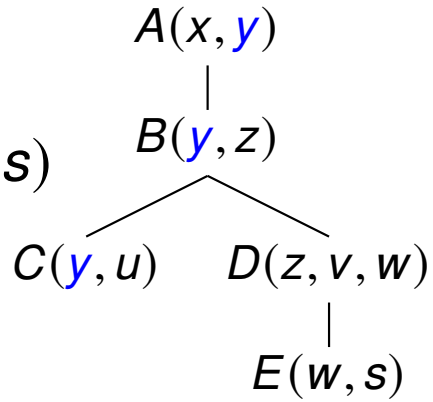
$$Q = A(x,y) \wedge B(y,z) \wedge C(y,u) \wedge D(z,v,w) \wedge E(w,s)$$



Examples

$Q = A(x,y) \wedge B(y,z) \wedge C(y,u) \wedge D(z,v,w) \wedge E(w,s)$

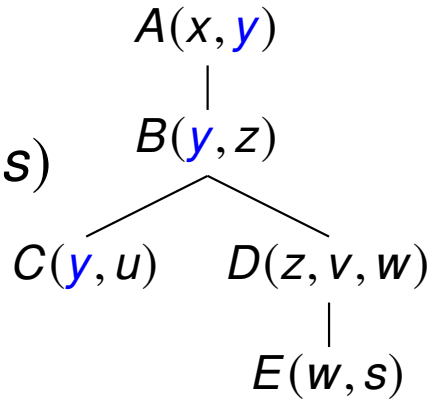
Acyclic. Eg. running intersection for y



Examples

$Q = A(x,y) \wedge B(y,z) \wedge C(y,u) \wedge D(z,v,w) \wedge E(w,s)$

Acyclic. Eg. running intersection for y

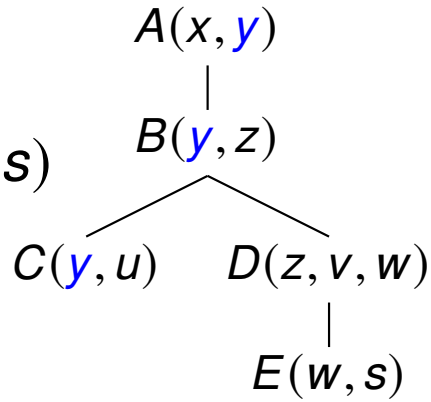


$Q = A(x,y) \wedge B(y,z) \wedge C(z,x)$

Examples

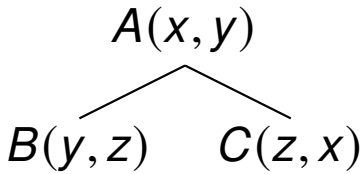
$Q = A(x,y) \wedge B(y,z) \wedge C(y,u) \wedge D(z,v,w) \wedge E(w,s)$

Acyclic. Eg. running intersection for y



$Q = A(x,y) \wedge B(y,z) \wedge C(z,x)$

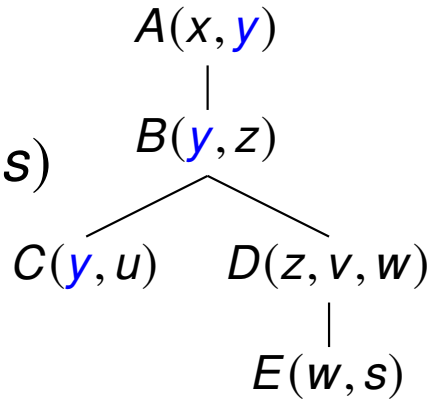
Cyclic; why?



Examples

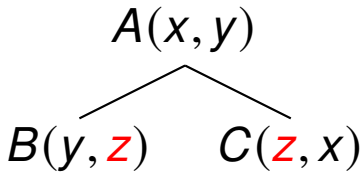
$Q = A(x,y) \wedge B(y,z) \wedge C(y,u) \wedge D(z,v,w) \wedge E(w,s)$

Acyclic. Eg. running intersection for y



$Q = A(x,y) \wedge B(y,z) \wedge C(z,x)$

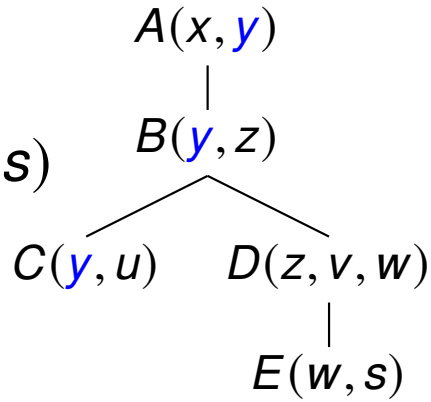
Cyclic; why? Running intersection for z



Examples

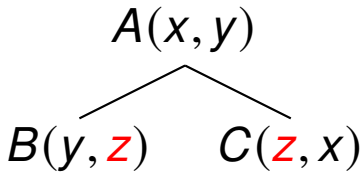
$Q = A(x,y) \wedge B(y,z) \wedge C(y,u) \wedge D(z,v,w) \wedge E(w,s)$

Acyclic. Eg. running intersection for y



$Q = A(x,y) \wedge B(y,z) \wedge C(z,x)$

Cyclic; why? Running intersection for z



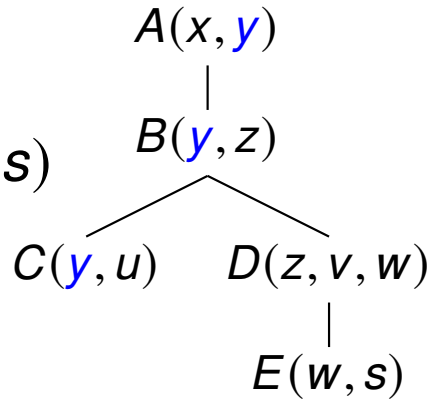
$Q = A(x,y) \wedge B(y,z) \wedge C(z,x) \wedge D(x,y,z)$

Acyclic?

Examples

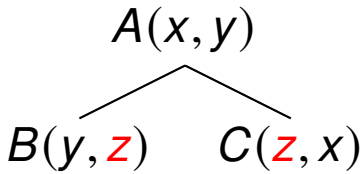
$Q = A(x,y) \wedge B(y,z) \wedge C(y,u) \wedge D(z,v,w) \wedge E(w,s)$

Acyclic. Eg. running intersection for y



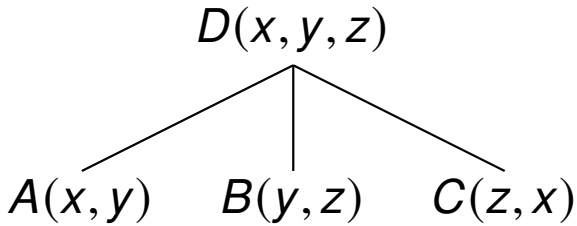
$Q = A(x,y) \wedge B(y,z) \wedge C(z,x)$

Cyclic; why? Running intersection for z



$Q = A(x,y) \wedge B(y,z) \wedge C(z,x) \wedge D(x,y,z)$

Acyclic? Yes



Discussion

A "join tree" is different from a "query plan" how?

The join tree may not be unique: $R(x, y) \wedge S(x, z) \wedge T(x, u)$.

Several definitions of acyclicity for hypergraphs [Fagin, 1983]

- α-acyclic – this is what we need
- β-acyclic
- γ-acyclic
- Berge-acyclic

Yannakakis' Algorithm

Overview

Introduced in [Yannakakis, 1981]

Computes an acyclic query in time $O(|\texttt{IN}| + |\texttt{OUT}|)$ (which is optimal), in the following cases:

- When Q is a Boolean query
- When Q is a full conjunctive query.
- When Q is "acyclic, free connex"

YA for a Boolean Query

Boolean, acyclic query $Q = R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n$, database D.

Compute $Q(D)$: the answer is either TRUE or FALSE: $|\texttt{OUT}| = 1$.

Fix a join tree T. Choose a root: the tree is now directed.

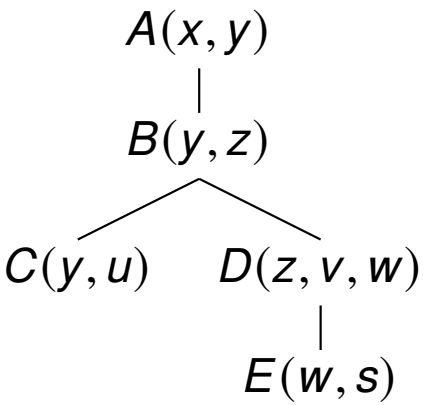
- Traverse the tree bottom-up and set $R_i := R_i \ltimes R_{\texttt{child}(i)}$.

$R_{\texttt{root}} = \emptyset$ then return FALSE. Otherwise return TRUE.

Runtime: $O(|\texttt{IN}|)$

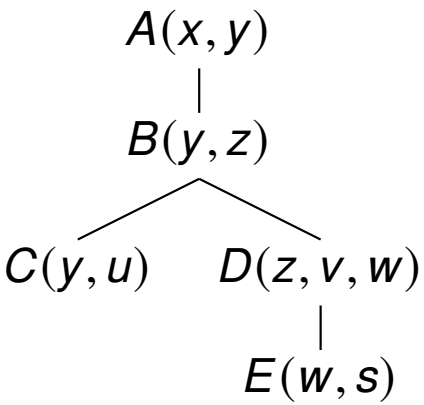
Example

$$Q = A(x,y) \wedge B(y,z) \wedge C(y,u) \wedge D(z,v,w) \wedge E(w,s)$$



Example

$$Q = A(x,y) \wedge B(y,z) \wedge C(y,u) \wedge D(z,v,w) \wedge E(w,s)$$



$$D := D \ltimes E$$

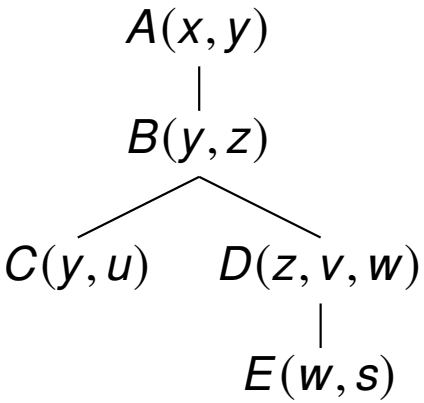
$$B := B \ltimes C$$

$$B := B \ltimes D$$

$$A := A \ltimes B$$

Example

$$Q = A(x,y) \wedge B(y,z) \wedge C(y,u) \wedge D(z,v,w) \wedge E(w,s)$$



$$D := D \ltimes E$$

$$B := B \ltimes C$$

$$B := B \ltimes D$$

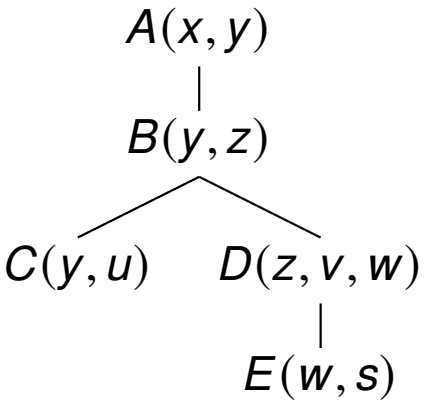
$$A := A \ltimes B$$

Correctness:

- $A \bowtie (\cdots) \neq \emptyset$ iff $A \ltimes (\cdots) \neq \emptyset$.

Example

$$Q = A(x,y) \wedge B(y,z) \wedge C(y,u) \wedge D(z,v,w) \wedge E(w,s)$$



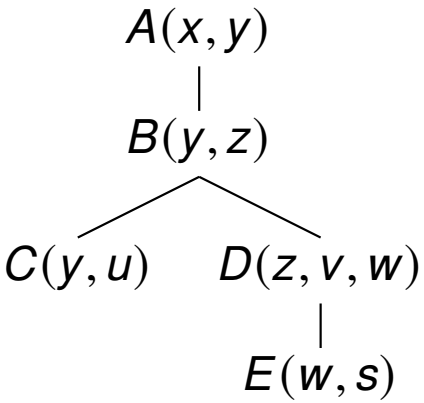
$$D := D \ltimes E$$
$$B := B \ltimes C$$
$$B := B \ltimes D$$
$$A := A \ltimes B$$

Correctness:

- $A \bowtie (\cdots) \neq \emptyset$ iff $A \ltimes (\cdots) \neq \emptyset$.
- $A \ltimes (B \bowtie (\cdots)) = A \ltimes (B \ltimes (\cdots))$ running intersection property.

Example

$$Q = A(x,y) \wedge B(y,z) \wedge C(y,u) \wedge D(z,v,w) \wedge E(w,s)$$



$$D := D \ltimes E$$
$$B := B \ltimes C$$
$$B := B \ltimes D$$
$$A := A \ltimes B$$

Correctness:

- $A \bowtie (\cdots) \neq \emptyset$ iff $A \ltimes (\cdots) \neq \emptyset$.
- $A \ltimes (B \bowtie (\cdots)) = A \ltimes (B \ltimes (\cdots))$ running intersection property.
- $B \bowtie (C \bowtie D \bowtie (\cdots)) = B \ltimes C \ltimes (D \bowtie \cdots)$
- Etc.

Semijoin Reduction

Query $Q = R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n$, database instance $\boldsymbol{D} = (R_1^D, \ldots, R_n^D)$

Q is semijoin reduced, or calibrated if $\forall i$: $R_i^D \subseteq \Pi_{\text{attrs}(R_i)}(Q(\boldsymbol{D}))$

Semijoin Reduction

Query $Q = R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n$, database instance $\boldsymbol{D} = (R_1^D, \ldots, R_n^D)$

Q is semijoin reduced, or calibrated if $\forall i$: $R_i^D \subseteq \Pi_{\text{attrs}(R_i)}(Q(\boldsymbol{D}))$

We also say that R_i^D does not have dangling tuples.

Semijoin Reduction

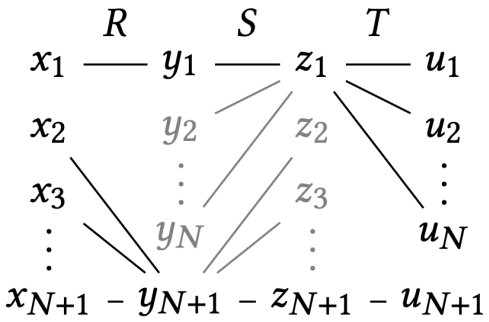
Query $Q = R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n$, database instance $\boldsymbol{D} = (R_1^D, \ldots, R_n^D)$

Q is semijoin reduced, or calibrated if $\forall i$: $R_i^D \subseteq \Pi_{\text{attrs}(R_i)}(Q(\boldsymbol{D}))$

We also say that R_i^D does not have dangling tuples.

What are the dangling tuples?
Compute the semijoin reduction

$$R(x,y) \bowtie S(y,z) \bowtie T(z,u)$$



(c) Database db_2

[Bekkers et al., 2025]

Semijoin Reduction

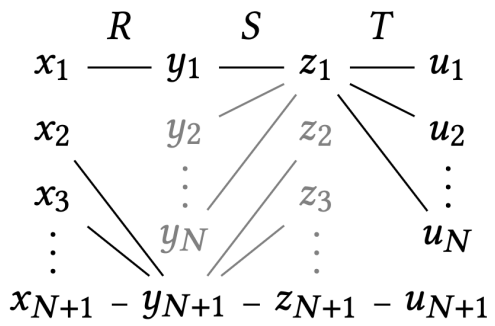
Query $Q = R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n$, database instance $\boldsymbol{D} = (R_1^D, \ldots, R_n^D)$

Q is semijoin reduced, or calibrated if $\forall i$: $R_i^D \subseteq \Pi_{\text{attrs}(R_i)}(Q(\boldsymbol{D}))$

We also say that R_i^D does not have dangling tuples.

What are the dangling tuples?
Compute the semijoin reduction

$$R(x,y) \bowtie S(y,z) \bowtie T(z,u)$$



(c) Database db_2

[Bekkers et al., 2025]

After reduction: $R^D = \{(x_1, y_1)\}, S^D = \{(y_1, z_1)\}, T^D = \{(z_1, u_1)\}$

Semijoin Reduction

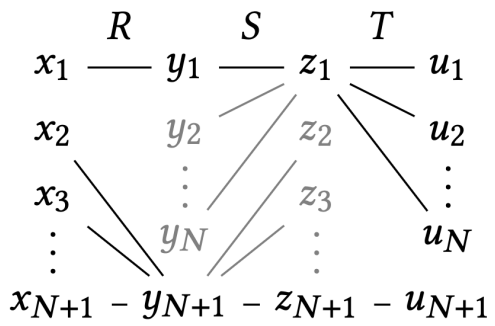
Query $Q = R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n$, database instance $\boldsymbol{D} = (R_1^D, \ldots, R_n^D)$

Q is semijoin reduced, or calibrated if $\forall i$: $R_i^D \subseteq \Pi_{\text{attrs}(R_i)}(Q(\boldsymbol{D}))$

We also say that R_i^D does not have dangling tuples.

What are the dangling tuples?
Compute the semijoin reduction

$$R(x,y) \bowtie S(y,z) \bowtie T(z,u)$$



(c) Database db_2

[Bekkers et al., 2025]

After reduction: $R^D = \{(x_1, y_1)\}, S^D = \{(y_1, z_1)\}, T^D = \{(z_1, u_1)\}$

Naive semijoin reduction algorithm: $\forall i$: $R_i := R_i \ltimes (R_1 \bowtie \cdots \bowtie R_n)$.

Semijoin Reduction Algorithm

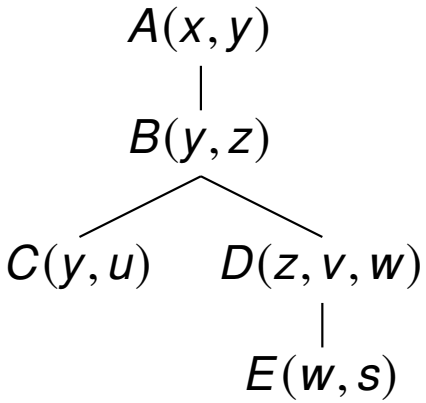
Goal: do semijoin reduction without computing the full query.

Variant of Yannakakis' algorithm for acyclic query Q.

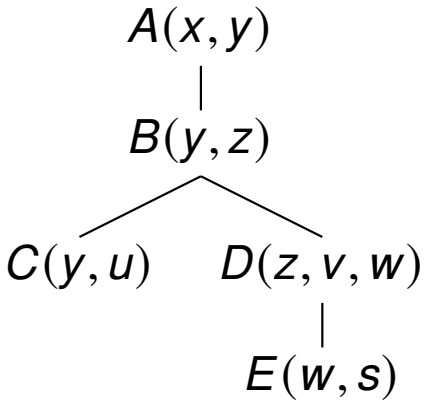
Semijoin Reduction.

- Traverse the tree bottom-up and set $R_i := R_i \ltimes R_{\texttt{child}(i)}$.
- Traverse the tree top-down and set $R_i := R_i \ltimes R_{\texttt{parent}(i)}$.

Example



Example



Semijoin Reduction
Bottom-up:

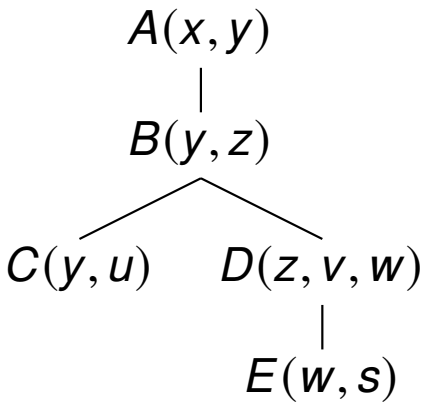
$$D := D \ltimes E$$

$$B := B \ltimes C$$

$$B := B \ltimes D$$

$$A := A \ltimes B$$

Example



Semijoin Reduction

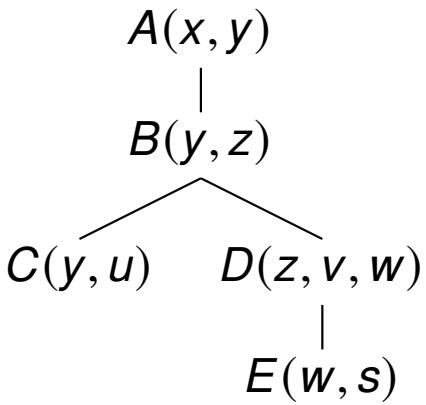
Bottom-up:

$$D := D \ltimes E$$
$$B := B \ltimes C$$
$$B := B \ltimes D$$
$$A := A \ltimes B$$

Top-down:

$$B := B \ltimes A$$
$$C := C \ltimes B$$
$$D := D \ltimes B$$
$$E := E \ltimes D$$

Example



Semijoin Reduction

Bottom-up:	Top-down:
$D := D \ltimes E$	$B := B \ltimes A$
$B := B \ltimes C$	$C := C \ltimes B$
$B := B \ltimes D$	$D := D \ltimes B$
$A := A \ltimes B$	$E := E \ltimes D$

Exercise*: prove correctness

Example:

$$B_{\text{final}}(y, z) \subseteq \Pi_{yz}(A(x, y) \bowtie \cdots)$$

where

$$B_{\text{final}} = (B \ltimes C \ltimes (D \ltimes E)) \ltimes A_{\text{final}}$$

Semijoin Reducing a Cyclic Query

$$A(x,y) \bowtie B(y,z) \bowtie C(z,x)$$

$$B := B \ltimes C$$

$$A := A \ltimes B$$

$$B := B \ltimes A$$

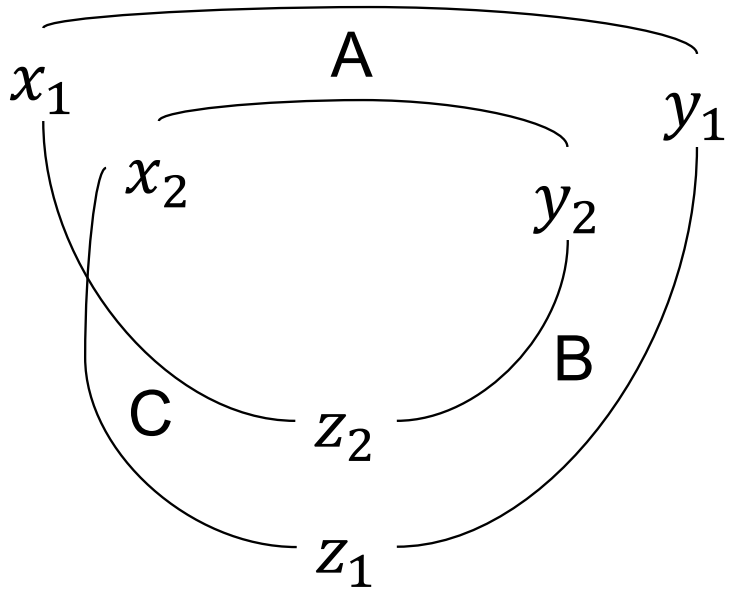
$$C := C \ltimes B$$

$$\ldots$$

Semijoin Reducing a Cyclic Query

$$A(x,y) \bowtie B(y,z) \bowtie C(z,x)$$

$$B := B \ltimes C$$
$$A := A \ltimes B$$
$$B := B \ltimes A$$
$$C := C \ltimes B$$
$$\ldots$$

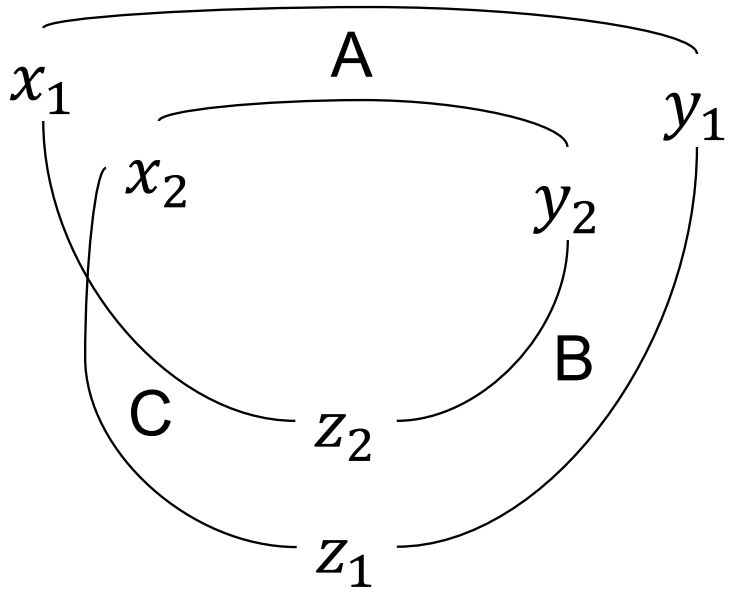


In class: What are the reduced relations? Does the reducer above work?

Semijoin Reducing a Cyclic Query

$$A(x,y) \bowtie B(y,z) \bowtie C(z,x)$$

$$\begin{aligned} B &:= B \ltimes C \\ A &:= A \ltimes B \\ B &:= B \ltimes A \\ C &:= C \ltimes B \\ &\dots \end{aligned}$$



In class: What are the reduced relations? Does the reducer above work?

Theorem A CQ admits a semijoin reducer iff it is acyclic.

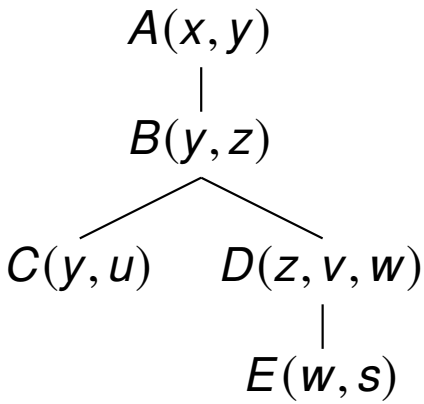
Full Conjunctive Query

Full CQ $Q(x_1, \ldots, x_m) = R_1 \bowtie \cdots \bowtie R_n$, database $\boldsymbol{D}$.
Fix a join tree T. Choose a root: the tree is now directed.

Full Conjunctive Query

Full CQ $Q(x_1, \ldots, x_m) = R_1 \bowtie \cdots \bowtie R_n$, database $\boldsymbol{D}$.
Fix a join tree T. Choose a root: the tree is now directed.

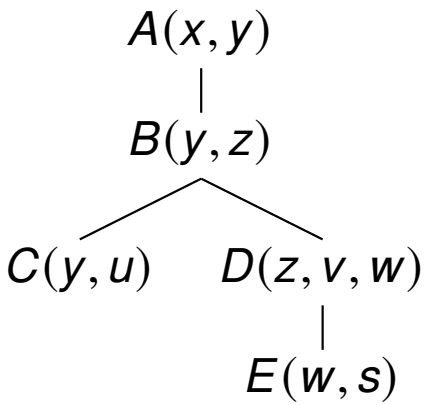
Naive computation: compute the joins top down in the tree.



Full Conjunctive Query

Full CQ $Q(x_1, \ldots, x_m) = R_1 \bowtie \cdots \bowtie R_n$, database $\boldsymbol{D}$.
Fix a join tree T. Choose a root: the tree is now directed.

Naive computation: compute the joins top down in the tree.

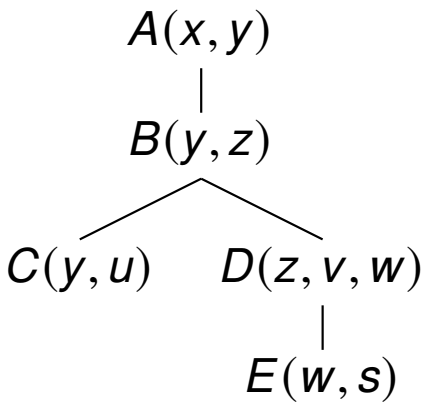


$$\begin{aligned}
\text{Out}_0 &:= \{()\} \\
\text{Out}_1 &:= \text{Out}_0 \bowtie A \\
\text{Out}_2 &:= \text{Out}_1 \bowtie B \\
\text{Out}_3 &:= \text{Out}_2 \bowtie C \\
\text{Out}_4 &:= \text{Out}_3 \bowtie D \\
Q &:= \text{Out}_4 \bowtie E
\end{aligned}$$

Full Conjunctive Query

Full CQ $Q(x_1, \ldots, x_m) = R_1 \bowtie \cdots \bowtie R_n$, database $\boldsymbol{D}$.
Fix a join tree T. Choose a root: the tree is now directed.

Naive computation: compute the joins top down in the tree.



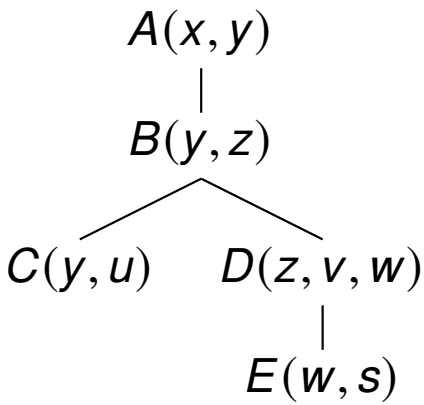
$$\begin{aligned}
\text{Out}_0 &:= \{()\} \\
\text{Out}_1 &:= \text{Out}_0 \bowtie A \\
\text{Out}_2 &:= \text{Out}_1 \bowtie B \\
\text{Out}_3 &:= \text{Out}_2 \bowtie C \\
\text{Out}_4 &:= \text{Out}_3 \bowtie D \\
Q &:= \text{Out}_4 \bowtie E
\end{aligned}$$

Side question: what is the query plan?

Full Conjunctive Query

Full CQ $Q(x_1, \ldots, x_m) = R_1 \bowtie \cdots \bowtie R_n$, database $\boldsymbol{D}$.
Fix a join tree T. Choose a root: the tree is now directed.

Naive computation: compute the joins top down in the tree.



$$\begin{aligned}
\text{Out}_0 &:= \{()\} \\
\text{Out}_1 &:= \text{Out}_0 \bowtie A \\
\text{Out}_2 &:= \text{Out}_1 \bowtie B \\
\text{Out}_3 &:= \text{Out}_2 \bowtie C \\
\text{Out}_4 &:= \text{Out}_3 \bowtie D \\
Q &:= \text{Out}_4 \bowtie E
\end{aligned}$$

Side question: what is the query plan?

Intermediate results may be $\gg |\texttt{OUT}|$; runtime $\gg O(|\texttt{IN}| + |\texttt{OUT}|)$.

YA for a Full Conjunctive Query

Full CQ Q, join tree T, database $\boldsymbol{D}$. Choose an arbitrary root in T.

Phase 1: Semijoin Reduction.

- Traverse the tree bottom-up and set $R_i := R_i \ltimes R_{\texttt{child}(i)}$.
- Traverse the tree top-down and set $R_i := R_i \ltimes R_{\texttt{parent}(i)}$.

YA for a Full Conjunctive Query

Full CQ Q, join tree T, database $\boldsymbol{D}$. Choose an arbitrary root in T.

Phase 1: Semijoin Reduction.

- Traverse the tree bottom-up and set $R_i := R_i \ltimes R_{\texttt{child}(i)}$.
- Traverse the tree top-down and set $R_i := R_i \ltimes R_{\texttt{parent}(i)}$.

Phase 2: Join Computation. Initialize $\text{Out}_0 := \{()\}$ (empty tuple).

- Traverse the tree top-down and set $\text{Out}_i := \text{Out}_{i-1} \bowtie R_i$.

Return Out_n.

Runtime: $O(|\texttt{IN}| + |\texttt{OUT}|)$

YA for a Full Conjunctive Query

Full CQ Q, join tree T, database $\boldsymbol{D}$. Choose an arbitrary root in T.

Phase 1: Semijoin Reduction.

- Traverse the tree bottom-up and set $R_i := R_i \ltimes R_{\texttt{child}(i)}$.
- ~~Traverse the tree top-down and set $R_i := R_i \ltimes R_{\texttt{parent}(i)}$.~~

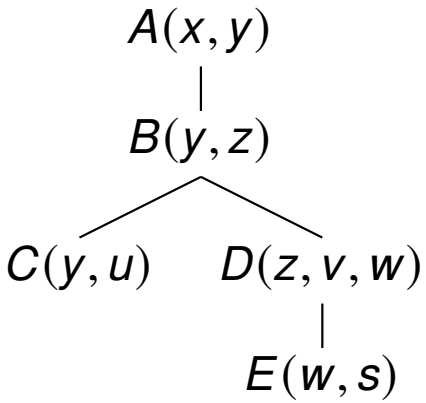
Phase 2: Join Computation. Initialize $\text{Out}_0 := \{()\}$ (empty tuple).

- Traverse the tree top-down and set $\text{Out}_i := \text{Out}_{i-1} \bowtie R_i$.

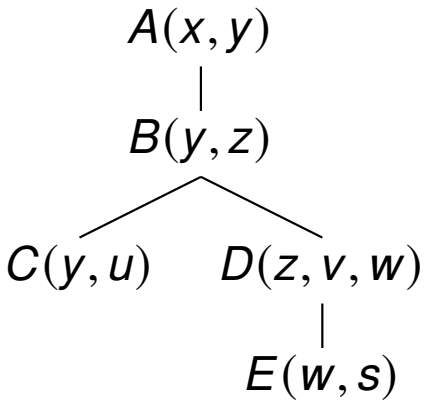
Return Out_n.

Runtime: $O(|\texttt{IN}| + |\texttt{OUT}|)$

Example



Example



Semijoin Reduction
Bottom-up:

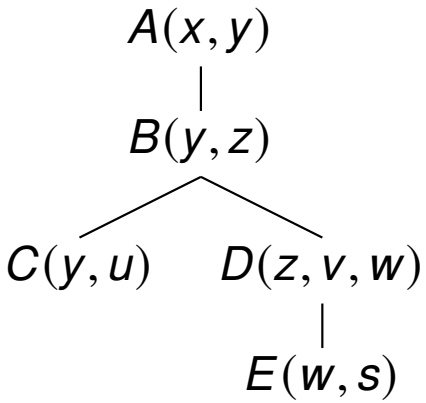
$$D := D \ltimes E$$

$$B := B \ltimes C$$

$$B := B \ltimes D$$

$$A := A \ltimes B$$

Example



Semijoin Reduction

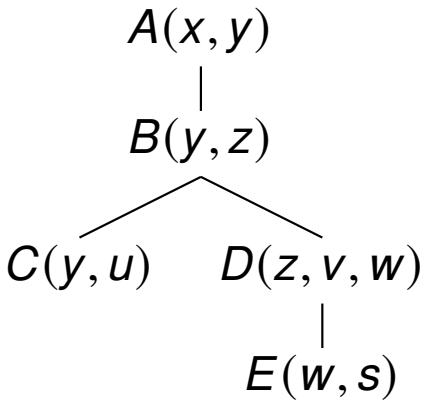
Bottom-up:

$$D := D \ltimes E$$
$$B := B \ltimes C$$
$$B := B \ltimes D$$
$$A := A \ltimes B$$

Top-down redundant:

$$B := B \ltimes A$$
$$C := C \ltimes B$$
$$D := D \ltimes B$$
$$E := E \ltimes D$$

Example



Semijoin Reduction

Bottom-up:

$$D := D \ltimes E$$
$$B := B \ltimes C$$
$$B := B \ltimes D$$
$$A := A \ltimes B$$

Top-down redundant:

$$B := B \ltimes A$$
$$C := C \ltimes B$$
$$D := D \ltimes B$$
$$E := E \ltimes D$$

Join Computation

$$\text{Out}_0 := \{()\}$$
$$\text{Out}_1 := \text{Out}_0 \bowtie A$$
$$\text{Out}_2 := \text{Out}_1 \bowtie B$$
$$\text{Out}_3 := \text{Out}_2 \bowtie C$$
$$\text{Out}_4 := \text{Out}_3 \bowtie D$$
$$Q := \text{Out}_4 \bowtie E$$

Discussion

- YA works only when the query is acyiclic.
  How do we check acyclicity? The GYO algorithm

- The runtime $O(|\texttt{IN}| + |\texttt{OUT}|)$ only holds for Boolean and Full queries.
  What other queries have this runtime? The Free Connex queries

The GYO Algorithm

Overview

We need a simple test to check whether Q is acyclic.

GYO algorithm is such a test.

Introduced independently by Graham, and by Yu and Özsoyoğlu: for history, see [Fagin, 1983].

Acyclic Query - GYO

The GYO Acyclicity Test (Graham and Yu-Oszoyoglu) Repeat:

- Remove an isolated variable (i.e. occurs in only one atom).
- Remove an ear (i.e. atom contain in another atom).

Q is a acyclic iff result is one empty edge.

Which var is isolated?

$$Q = A(x,y) \wedge B(y,z) \wedge C(y,u) \wedge D(z,v,w) \wedge E(w,s)$$

Acyclic Query - GYO

The GYO Acyclicity Test (Graham and Yu-Oszoyoglu) Repeat:

- Remove an isolated variable (i.e. occurs in only one atom).
- Remove an ear (i.e. atom contain in another atom).

Q is a acyclic iff result is one empty edge.

Which var is isolated? Answer: x : $Q = A(x,y) \wedge B(y,z) \wedge C(y,u) \wedge D(z,v,w) \wedge E(w,s)$

Acyclic Query - GYO

The GYO Acyclicity Test (Graham and Yu-Oszoyoglu) Repeat:

- Remove an isolated variable (i.e. occurs in only one atom).
- Remove an ear (i.e. atom contain in another atom).

Q is a acyclic iff result is one empty edge.

$$Q = A(x,y) \wedge B(y,z) \wedge C(y,u) \wedge D(z,v,w) \wedge E(w,s)$$

Which atom is an ear? $\rightarrow A(y) \wedge B(y,z) \wedge C(y,u) \wedge D(z,v,w) \wedge E(w,s)$

Acyclic Query - GYO

The GYO Acyclicity Test (Graham and Yu-Oszoyoglu) Repeat:

- Remove an isolated variable (i.e. occurs in only one atom).
- Remove an ear (i.e. atom contain in another atom).

Q is a acyclic iff result is one empty edge.

$$Q = A(x,y) \wedge B(y,z) \wedge C(y,u) \wedge D(z,v,w) \wedge E(w,s)$$

Which atom is an ear? $\rightarrow A(y) \wedge B(y,z) \wedge C(y,u) \wedge D(z,v,w) \wedge E(w,s)$

Acyclic Query - GYO

The GYO Acyclicity Test (Graham and Yu-Oszoyoglu) Repeat:

- Remove an isolated variable (i.e. occurs in only one atom).
- Remove an ear (i.e. atom contain in another atom).

Q is a acyclic iff result is one empty edge.

$$
\begin{aligned}
Q =& A(x,y) \wedge B(y,z) \wedge C(y,u) \wedge D(z,v,w) \wedge E(w,s) \\
\rightarrow& A(y) \wedge B(y,z) \wedge C(y,u) \wedge D(z,v,w) \wedge E(w,s) \\
\rightarrow& B(y,z) \wedge C(y,u) \wedge D(z,v,w) \wedge E(w,s) \\
\rightarrow& B(y,z) \wedge C(y) \wedge D(z,w) \wedge E(w) \\
\rightarrow& B(y,z) \wedge D(z,w) \\
\rightarrow& B(z) \wedge D(z) \\
\rightarrow& D(z) \\
\rightarrow& - \quad \text{Acyclic!}
\end{aligned}
$$

Acyclic Query - GYO

The GYO Acyclicity Test (Graham and Yu-Oszoyoglu) Repeat:

- Remove an isolated variable (i.e. occurs in only one atom).
- Remove an ear (i.e. atom contain in another atom).

Q is a acyclic iff result is one empty edge.

$$
\begin{aligned}
Q =& A(x,y) \wedge B(y,z) \wedge C(y,u) \wedge D(z,v,w) \wedge E(w,s) \\
\rightarrow& A(y) \wedge B(y,z) \wedge C(y,u) \wedge D(z,v,w) \wedge E(w,s) \\
\rightarrow& B(y,z) \wedge C(y,u) \wedge D(z,v,w) \wedge E(w,s) \\
\rightarrow& B(y,z) \wedge C(y) \wedge D(z,w) \wedge E(w) \\
\rightarrow& B(y,z) \wedge D(z,w) \\
\rightarrow& B(z) \wedge D(z) \\
\rightarrow& D(z) \\
\rightarrow& - \quad \text{Acyclic!}
\end{aligned}
$$

The order in which we eliminate variables is called a variable elimination order
In our example: x, u, v, s, y, w, z

Discussion

In class: we can recover the join tree from the variable elimination order. This principle also applies to tree decompositions, which we will discuss in a few lectures.

Free Connex Queries

Motivation

Yannakakis' algorithms runs in time $O(|\texttt{IN}| + |\texttt{OUT}|)$ in two cases:

- Boolean Query: $|\texttt{OUT}| = 1$.
- Full Conjunctive query $|\texttt{OUT}|$ is largest.

What is we have some output variables, but not all?

General CQ

$$Q(x_{i_1}, \ldots, x_{i_p}) = R_1 \bowtie \cdots \bowtie R_n$$

Naive approach: compute full query $Q(x_1, \ldots, x_n)$, project on head vars.
Runtime: $O(|\mathtt{IN}| + |\mathtt{OUT}_{\text{full}}|) \gg O(|\text{IN}| + |\text{OUT}|)$

General CQ

$$Q(x_{i_1}, \ldots, x_{i_p}) = R_1 \bowtie \cdots \bowtie R_n$$

Naive approach: compute full query $Q(x_1, \ldots, x_n)$, project on head vars.
Runtime: $O(|\texttt{IN}| + |\texttt{OUT}_{\text{full}}|) \gg O(|\text{IN}| + |\text{OUT}|)$

Can't do much better:
The Boolean matrix multiplication conjecture if A, B are $N \times N$ Boolean matrices, then there exists no algorithm for computing $A \cdot B$ in times $O(N^2)$.

Can't compute $Q(i,k) = \exists j(A(i,j) \wedge B(j,k))$ in time $O(|\text{IN}| + |\text{OUT}|)$.

General CQ

$$Q(x_{i_1}, \ldots, x_{i_p}) = R_1 \bowtie \cdots \bowtie R_n$$

Naive approach: compute full query $Q(x_1, \ldots, x_n)$, project on head vars.
Runtime: $O(|\texttt{IN}| + |\texttt{OUT}_{\text{full}}|) \gg O(|\text{IN}| + |\text{OUT}|)$

Can't do much better:
The Boolean matrix multiplication conjecture if A, B are $N \times N$ Boolean matrices, then there exists no algorithm for computing $A \cdot B$ in times $O(N^2)$.

Can't compute $Q(i,k) = \exists j(A(i,j) \wedge B(j,k))$ in time $O(|\text{IN}| + |\text{OUT}|)$.

Standard approach: in phase 2 keep all outputs vars.
Runtime: $O(|\texttt{IN}| \cdot |\texttt{OUT}|)$

Free Connex Query

$$Q(x_{i_1}, \ldots, x_{i_p}) = R_1 \bowtie \cdots \bowtie R_n$$

Q is acyclic free-connex if it is acyclic, and remains acyclic after we add an atom $\text{Out}(x_{i_1}, \ldots, x_{i_p})$.

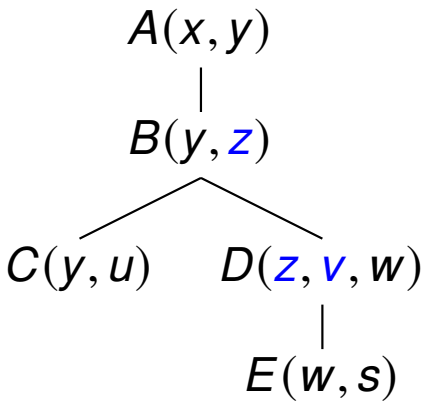
Equivalently: there exists a variable elimination order where $x_{i_1}, \ldots, x_{i_p}$ are the last variables.

Yannakakis' algorithm: in Phase 2, compute the joins towards the new atom $\text{Out}(x_1, \ldots, x_p)$.

Runtime: $O(|\texttt{IN}| + |\texttt{OUT}|)$

Example of a Free-Connex Query

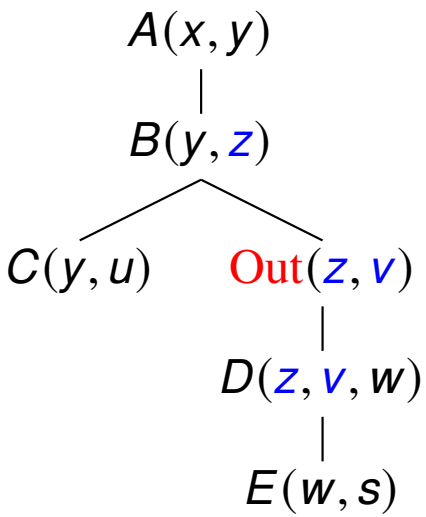
$Q(z, v) =$



Where do we place
Out(z, v)?

Example of a Free-Connex Query

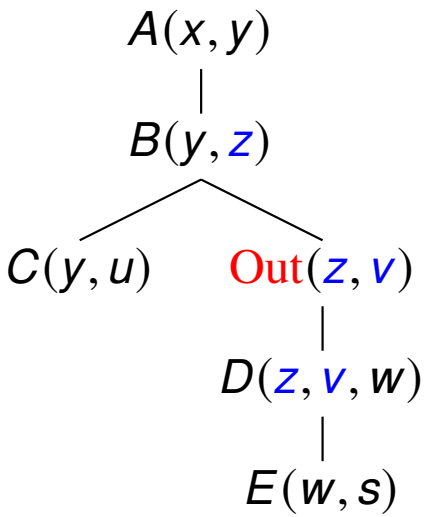
$Q(z, v) =$



Where do we place $\text{Out}(z, v)$?

Example of a Free-Connex Query

$Q(z, v) =$

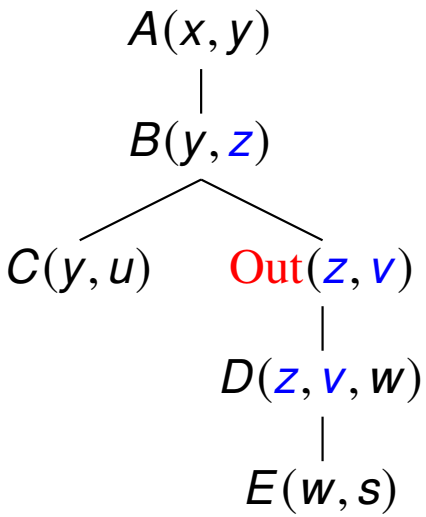


Semijoin Reduction

As before.

Example of a Free-Connex Query

$Q(z, v) =$



Join Computation

$$
\begin{aligned}
T_1(y) &:= A(x, y) \\
T_2(y, z) &:= T_1(y) \bowtie B(y, z) \\
T_3(y) &:= C(y, u) \\
T_4(z) &:= T_2(y, z) \bowtie T_3(y) \\
T_5(w) &:= E(w, s) \\
T_6(z, v) &:= T_5(w) \bowtie D(z, v, w) \\
T_7(z, v) &:= T_6(z, v) \bowtie T_4(z)
\end{aligned}
$$

Return $T_7(z, v)$.

Semijoin Reduction

As before.

The tree traversal is from the leaves towards Out(z, v).

Each T_i is either a subset of some input relation, or of the output $Q(z, v)$, hence Time= $O(|\text{IN}| + |\text{OUT}|)$

Summary

- Yannakakis' algorithm: Semijoin reduction (up, then down), then joins.
- Most SQL queries in practice are acyclic.
- In class Why don't database engines run Yannakakis algorithm?

Bekkers, L., Neven, F., Vansummeren, S., and Wang, Y. R. (2025).

Instance-optimal acyclic join processing without regret: Engineering the yannakakis algorithm in column stores.
Proc. VLDB Endow., 18(8):2413–2426.

Fagin, R. (1983).

Degrees of acyclicity for hypergraphs and relational database schemes.
J. ACM, 30(3):514–550.

Yannakakis, M. (1981).

Algorithms for acyclic database schemes.
In Very Large Data Bases, 7th International Conference, September 9-11, 1981, Cannes, France, Proceedings, pages 82–94. IEEE Computer Society.