

Topic 3: Efficient query evaluation

Unit 1: Acyclic query evaluation

Lecture 18

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp22)

<https://northeastern-datalab.github.io/cs7240/sp22/>

3/25/2022

Pre-class conversations

- Suggestion: Scribes with 2 iterations
- We continue with our example from last time
- Today:
 - Acyclic queries, Yannakakis, Hypergraphs, GYO reduction, Semi-join reducers
 - Possibly already: cycles

Symmetries and Complexity

Andrei A. Bulatov   

School of Computing Science, Simon Fraser University, Burnaby, Canada

Abstract

The Constraint Satisfaction Problem (CSP) and a number of problems related to it have seen major advances during the past three decades. In many cases the leading driving force that made these advances possible has been the so-called algebraic approach that uses symmetries of constraint problems and tools from algebra to determine the complexity of problems and design solution algorithms. In this presentation we give a high level overview of the main ideas behind the algebraic approach illustrated by examples ranging from the regular CSP, to counting problems, to optimization and promise problems, to graph isomorphism.

April 1, 12pm, Paper discussion @ WVH 462

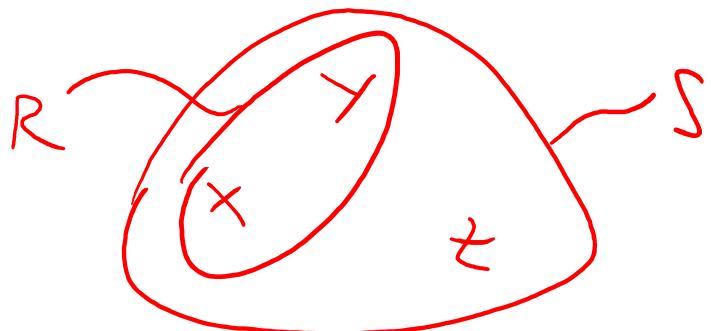
GYO reduction: Example 2



- GYO Ear removal

$$Q :- R(x,y,z), S(y,p), T(y,z,p), U(z,u,p), W(u,p,w).$$

- remove ears (= edges)
 - remove isolated nodes (variables)
 - remove consumed or empty edges (atoms)



Join tree

?

Query hypergraph

?

GYO reduction: Example 2



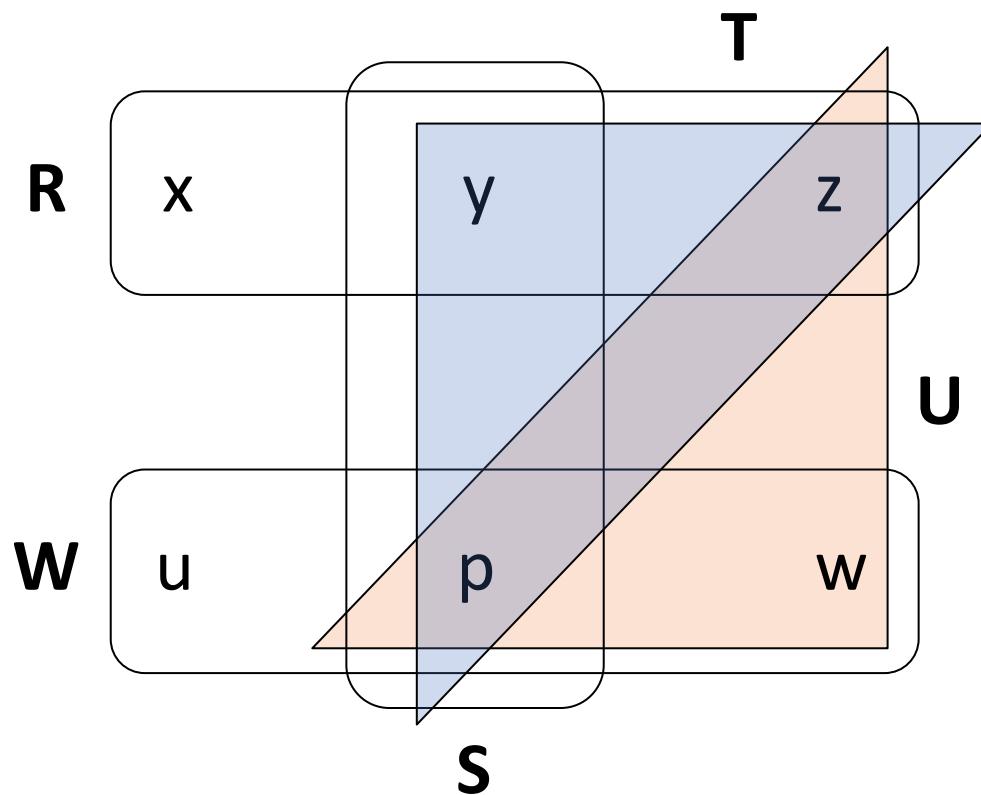
- GYO Ear removal

$$Q :- R(x,y,z), S(y,p), T(y,z,p), U(z,u,p), W(u,p,w).$$

- remove ears (= edges)
 - remove isolated nodes (variables)
 - remove consumed or empty edges (atoms)

Join tree

?



GYO reduction: Example 2

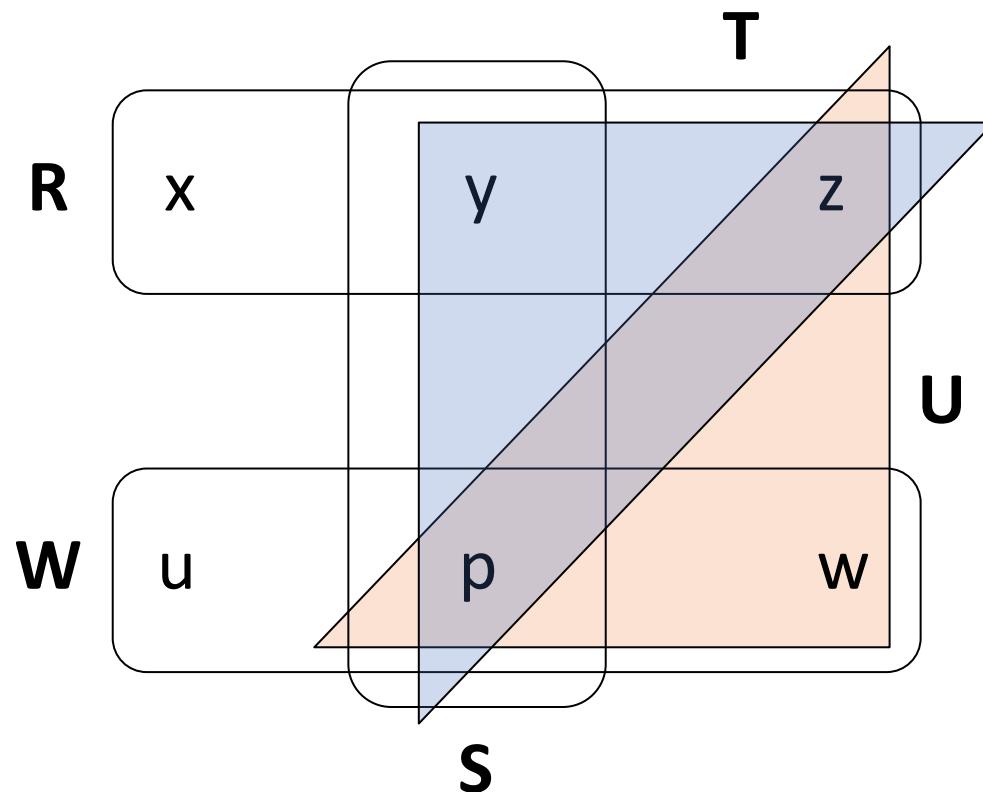
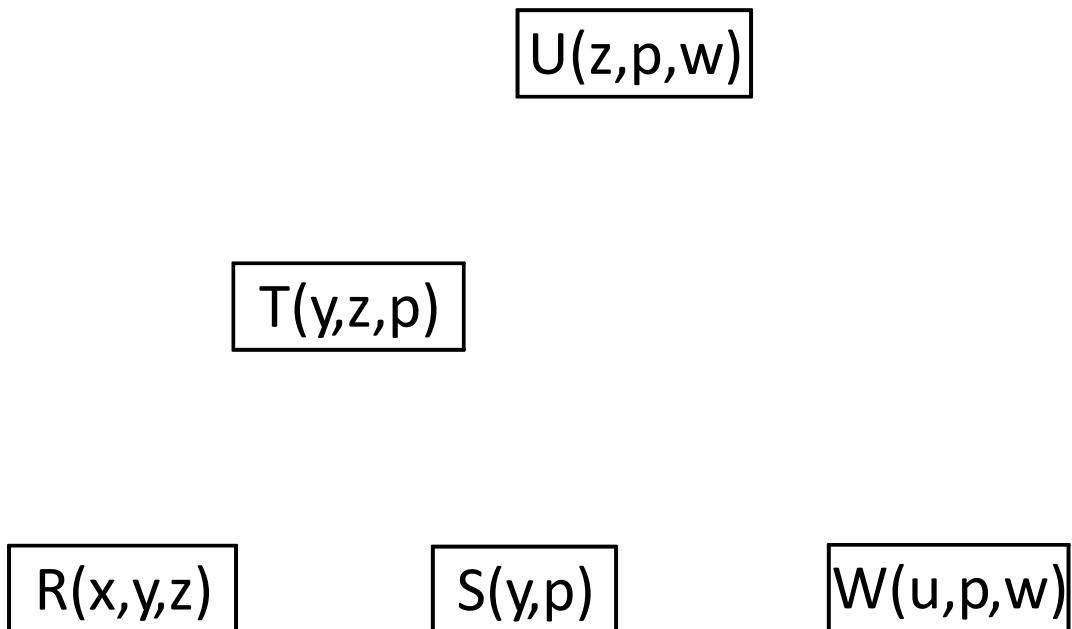


- GYO Ear removal

$$Q :- R(x,y,z), S(y,p), T(y,z,p), U(z,u,p), W(u,p,w).$$

- remove ears (= edges)

- remove isolated nodes (variables)
- remove consumed or empty edges (atoms)





GYO reduction: Example 2

- GYO Ear removal

$Q :- R(x,y,z), S(y,p), T(y,z,p), U(z,u,p), W(u,p,w).$

- remove ears (= edges)
 - remove isolated nodes (variables)
 - remove consumed or empty edges (atoms)

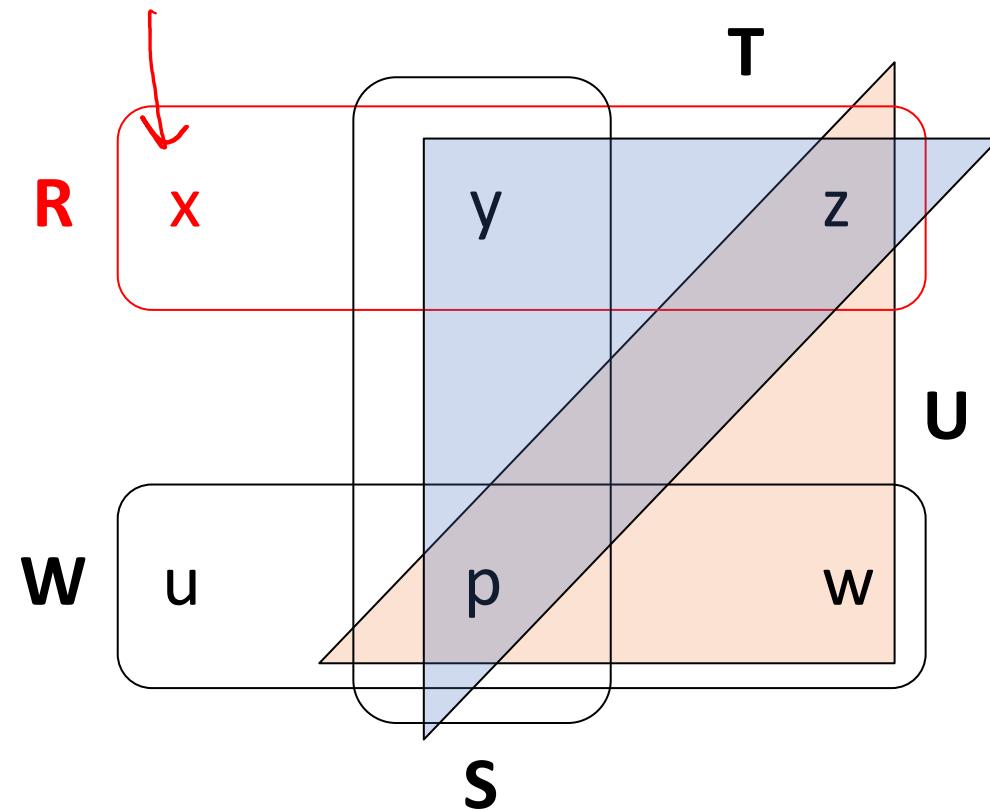
$U(z,p,w)$

$T(y,z,p)$

$R(x,y,z)$

$S(y,p)$

$W(u,p,w)$



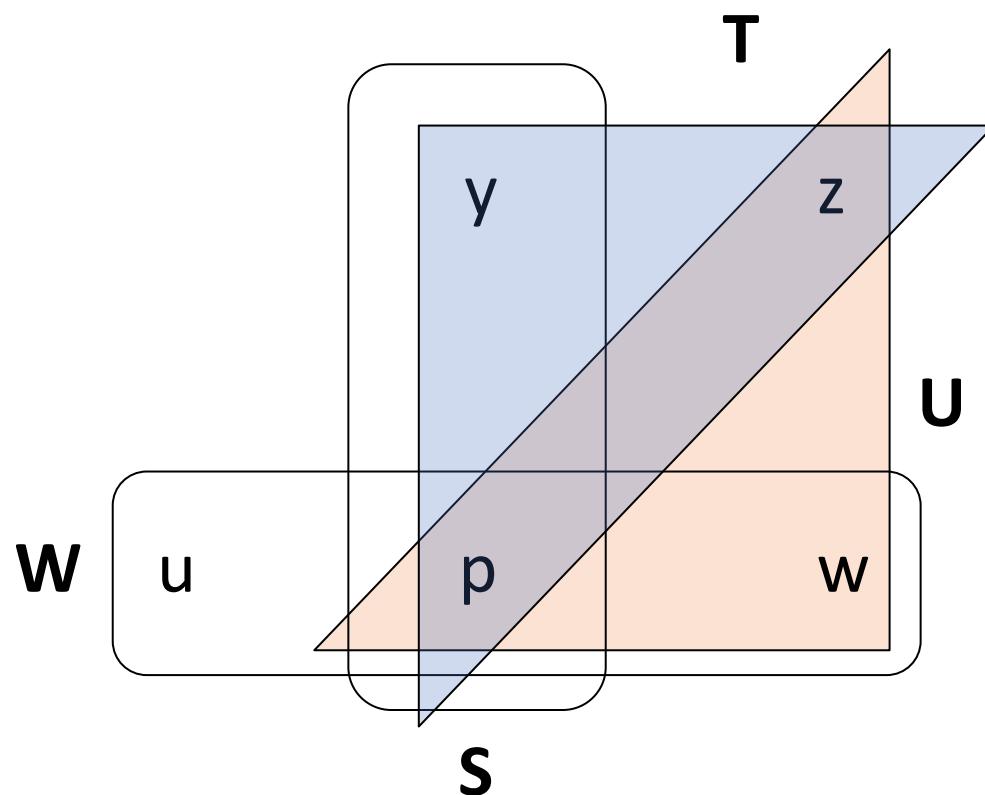
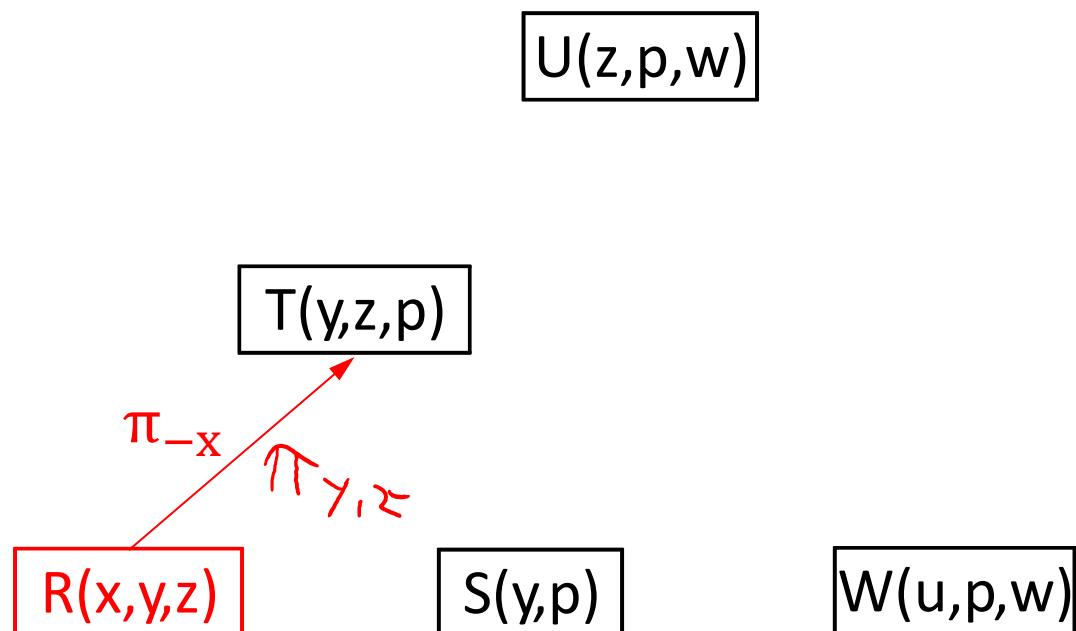
GYO reduction: Example 2



- GYO Ear removal

$$Q :- R(x,y,z), S(y,p), T(y,z,p), U(z,u,p), W(u,p,w).$$

- remove ears (= edges)
 - remove isolated nodes (variables)
 - remove consumed or empty edges (atoms)



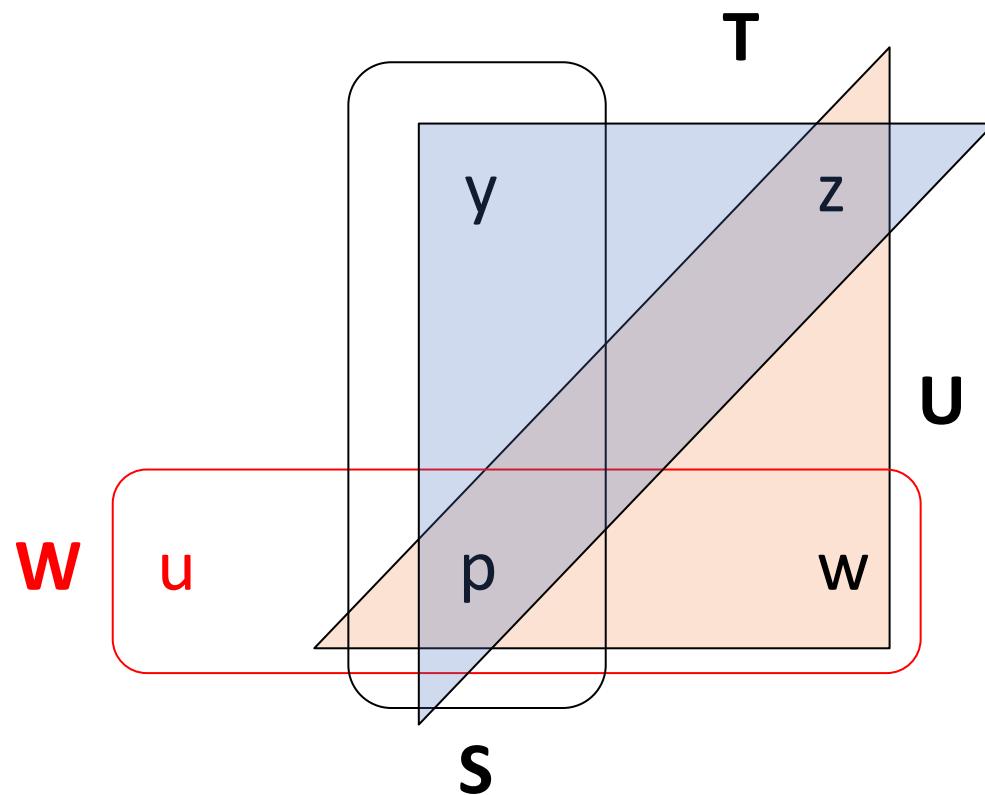
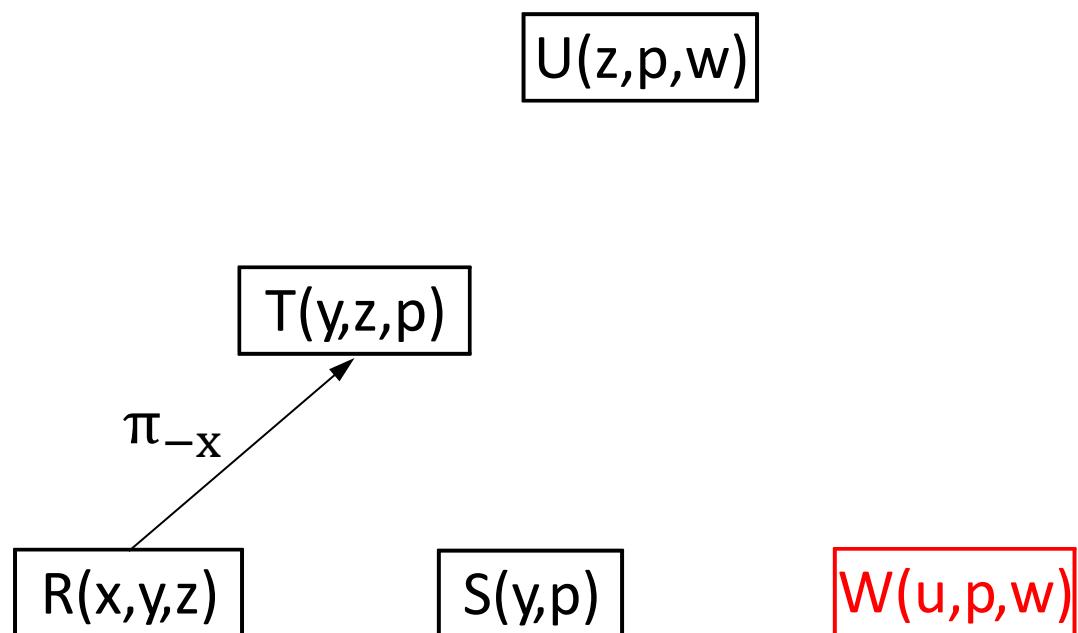
GYO reduction: Example 2



- GYO Ear removal

$$Q :- R(x,y,z), S(y,p), T(y,z,p), U(z,u,p), W(u,p,w).$$

- remove ears (= edges)
 - remove isolated nodes (variables)
 - remove consumed or empty edges (atoms)



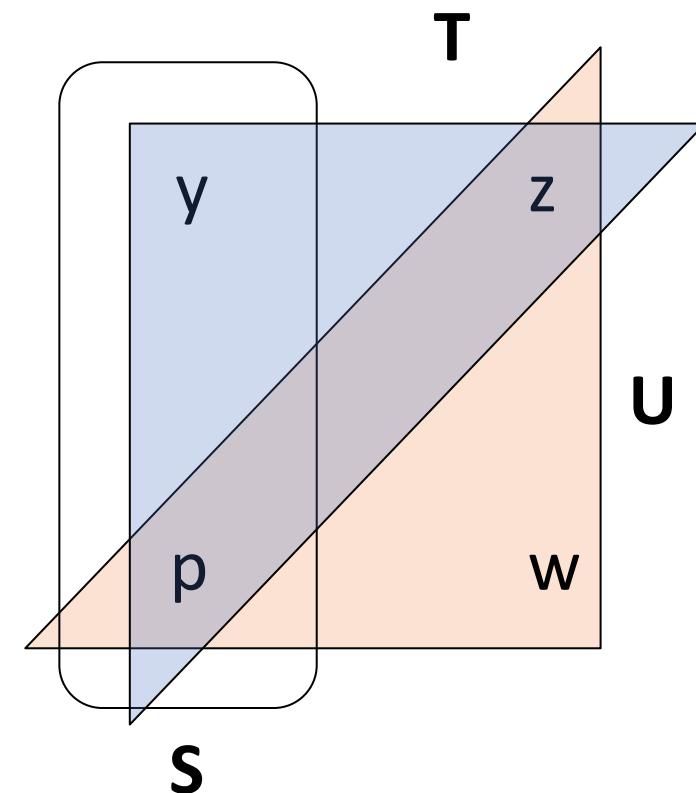
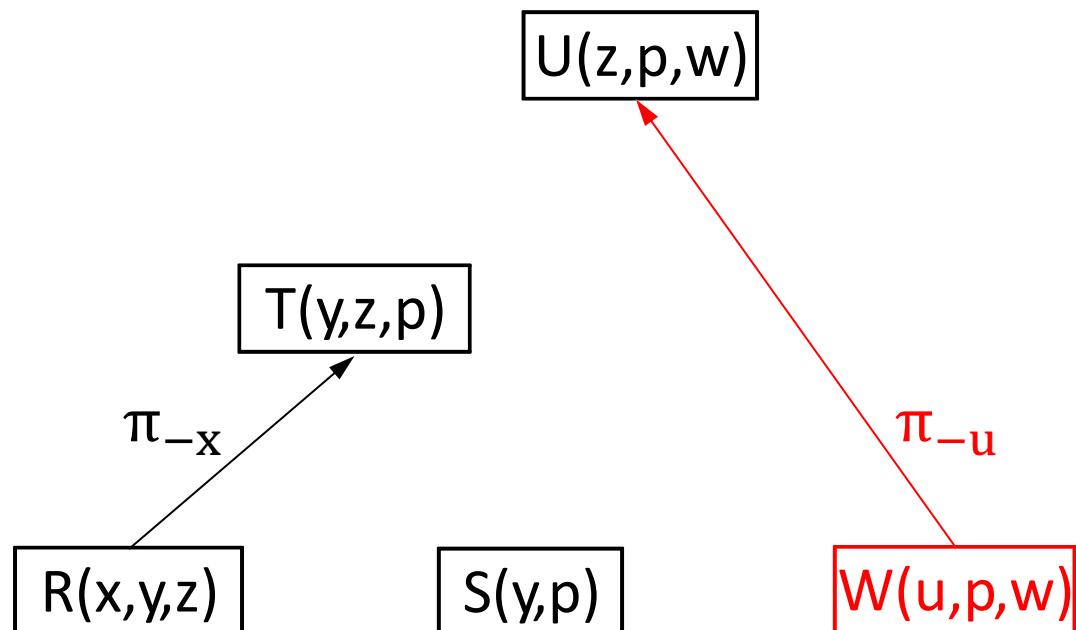
GYO reduction: Example 2



- GYO Ear removal

$$Q :- R(x,y,z), S(y,p), T(y,z,p), U(z,u,p), W(u,p,w).$$

- remove ears (= edges)
 - remove isolated nodes (variables)
 - remove consumed or empty edges (atoms)



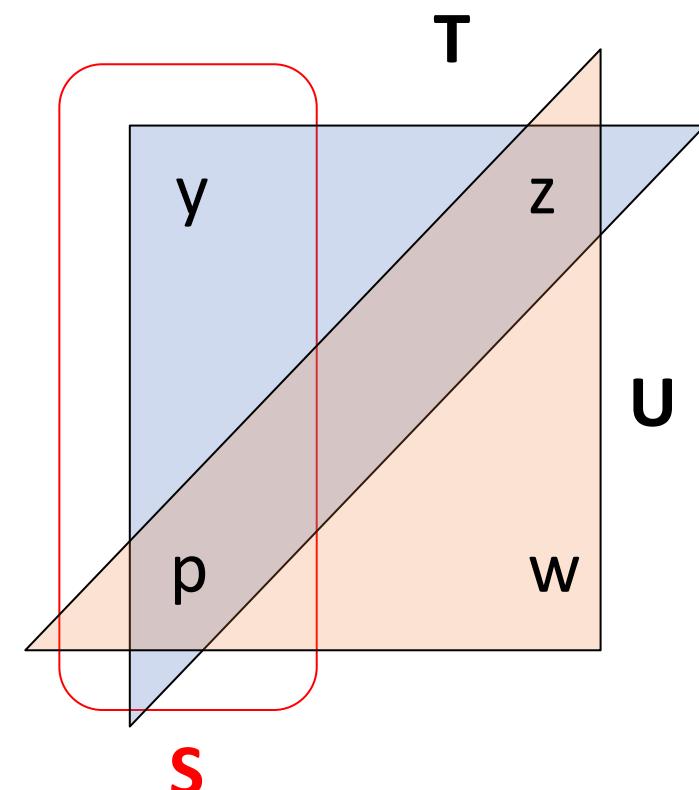
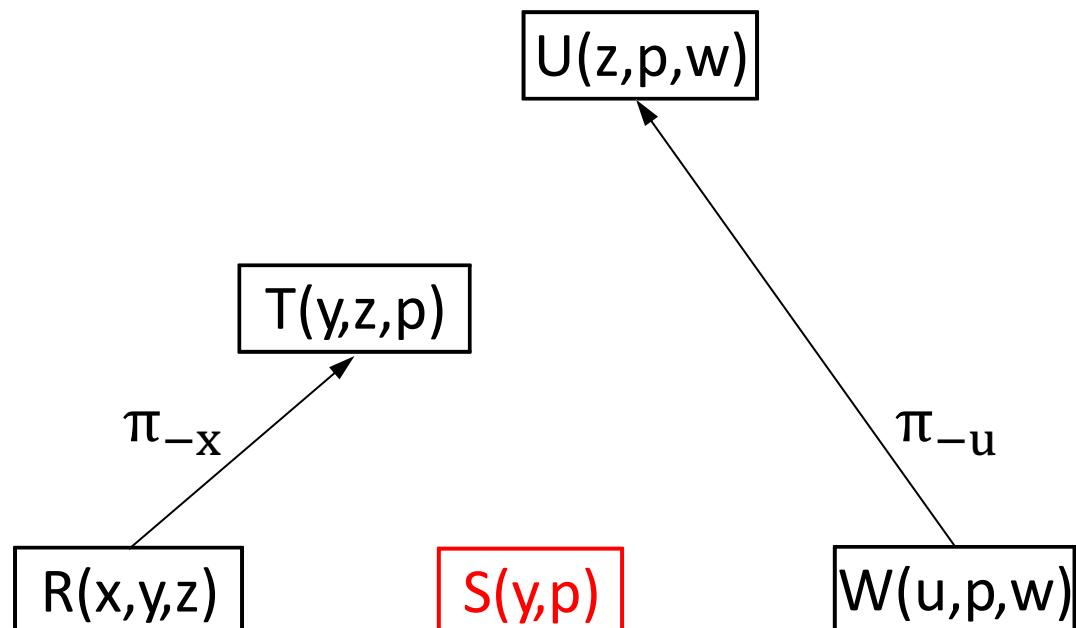
GYO reduction: Example 2



- GYO Ear removal

$$Q :- R(x,y,z), S(y,p), T(y,z,p), U(z,u,p), W(u,p,w).$$

- remove ears (= edges)
 - remove isolated nodes (variables)
 - remove consumed or empty edges (atoms)



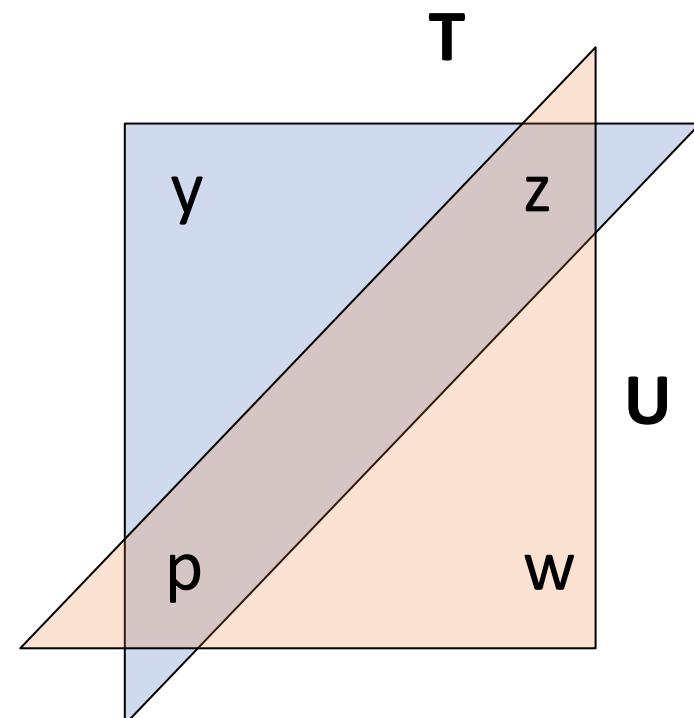
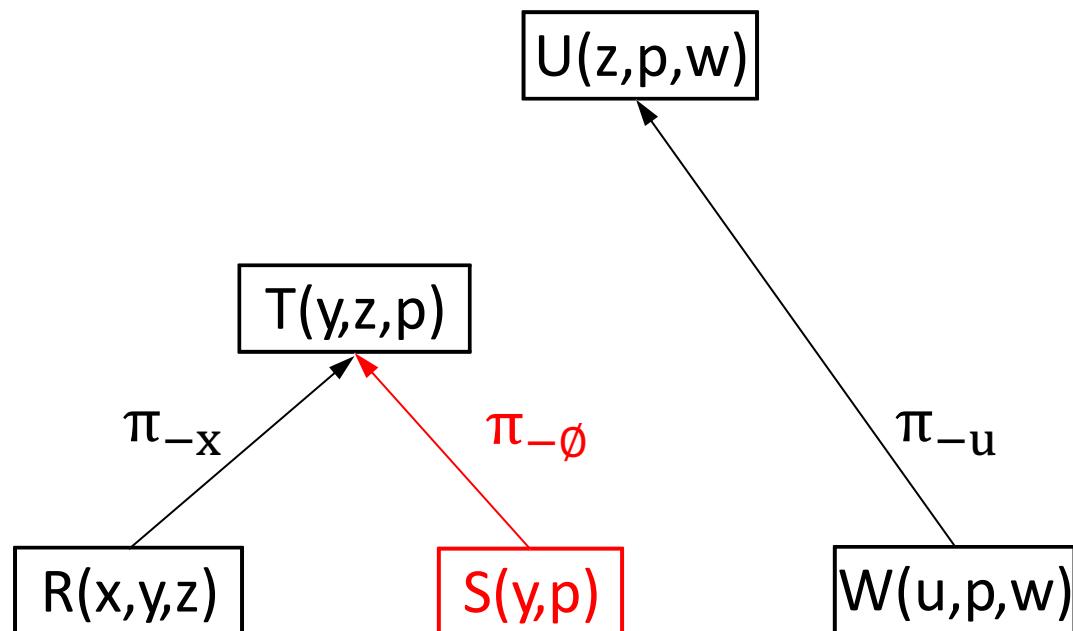
GYO reduction: Example 2



- GYO Ear removal

$$Q :- R(x,y,z), S(y,p), T(y,z,p), U(z,u,p), W(u,p,w).$$

- remove ears (= edges)
 - remove isolated nodes (variables)
 - remove consumed or empty edges (atoms)



GYO reduction: Example 2

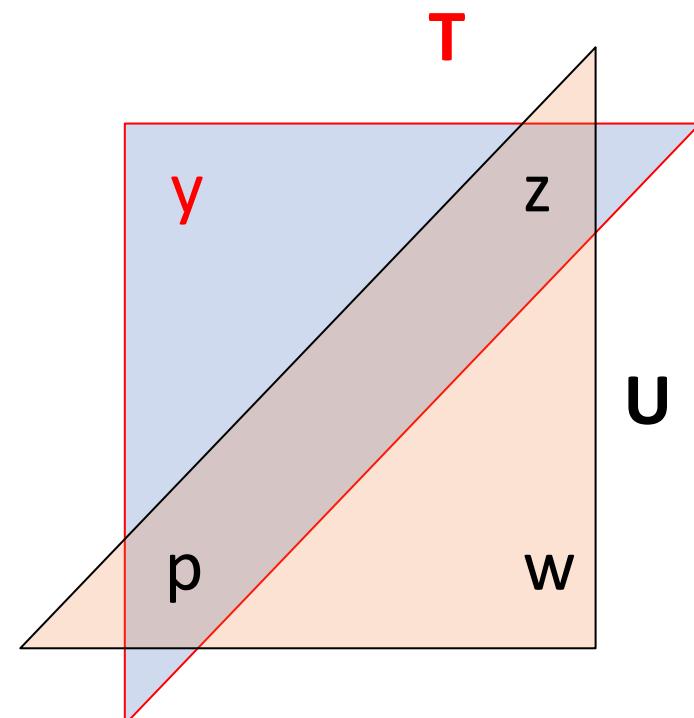
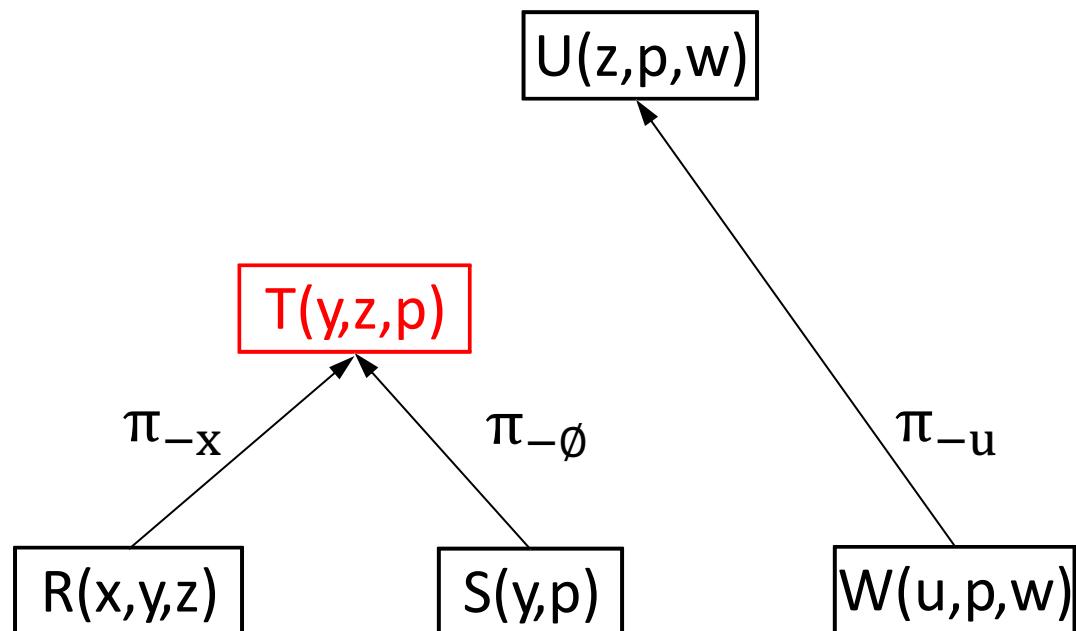


- GYO Ear removal

$$Q :- R(x,y,z), S(y,p), T(y,z,p), U(z,u,p), W(u,p,w).$$

- remove ears (= edges)

- remove isolated nodes (variables)
- remove consumed or empty edges (atoms)



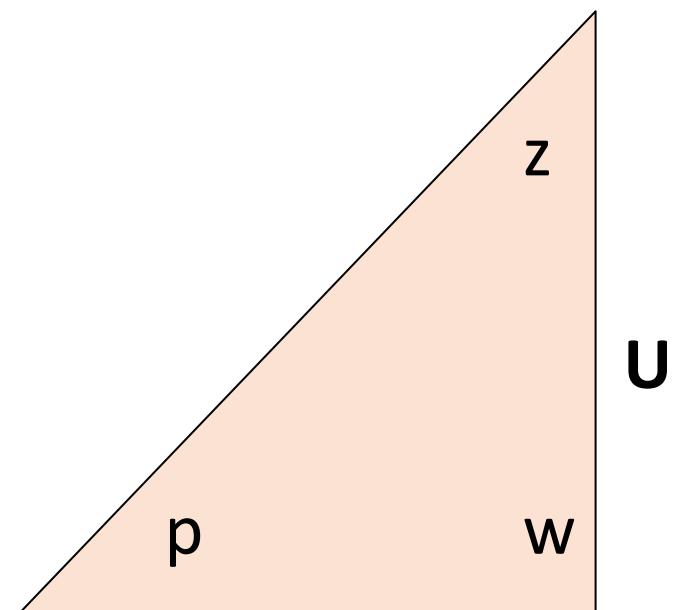
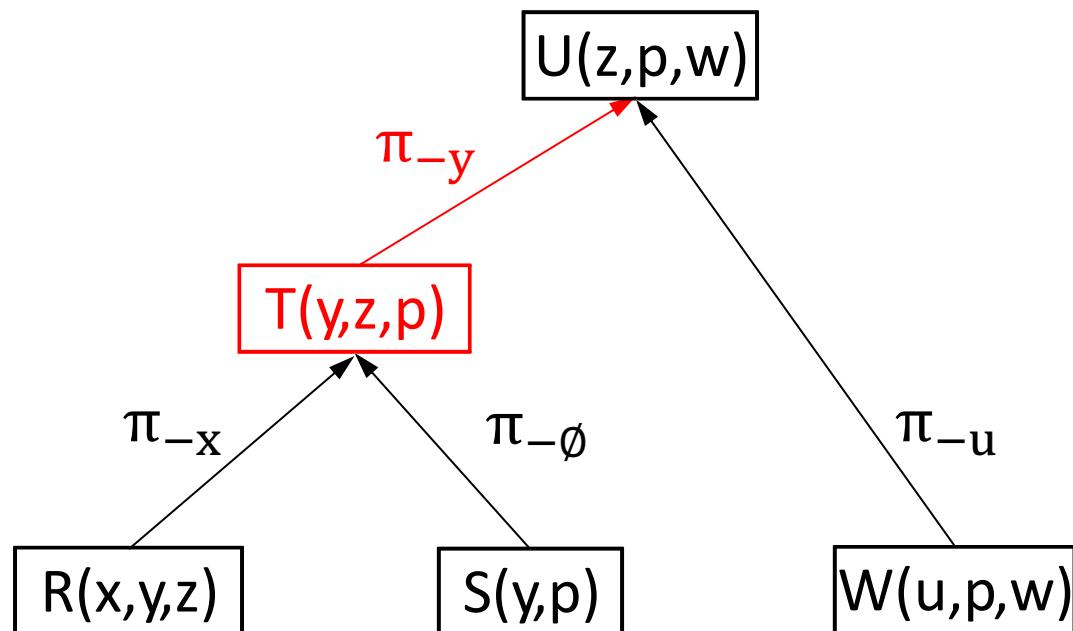
GYO reduction: Example 2



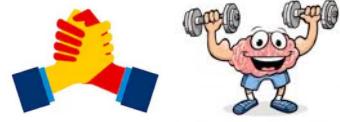
- GYO Ear removal

$$Q :- R(x,y,z), S(y,p), T(y,z,p), U(z,u,p), W(u,p,w).$$

- remove ears (= edges)
 - remove isolated nodes (variables)
 - remove consumed or empty edges (atoms)



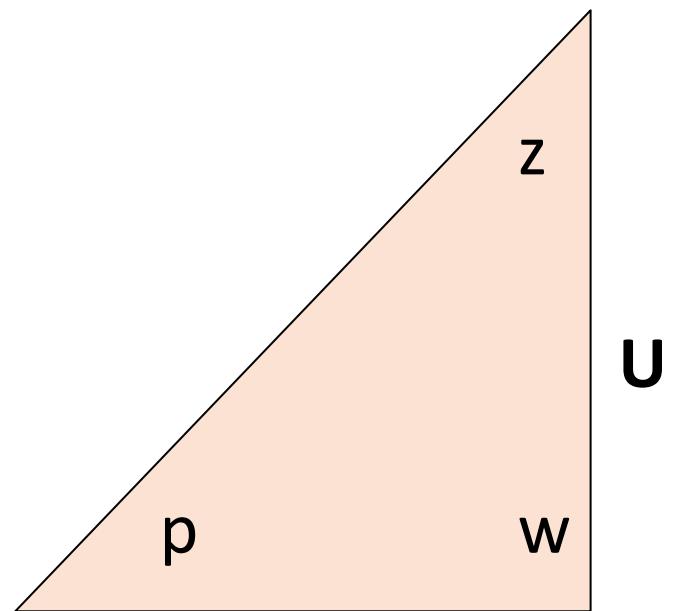
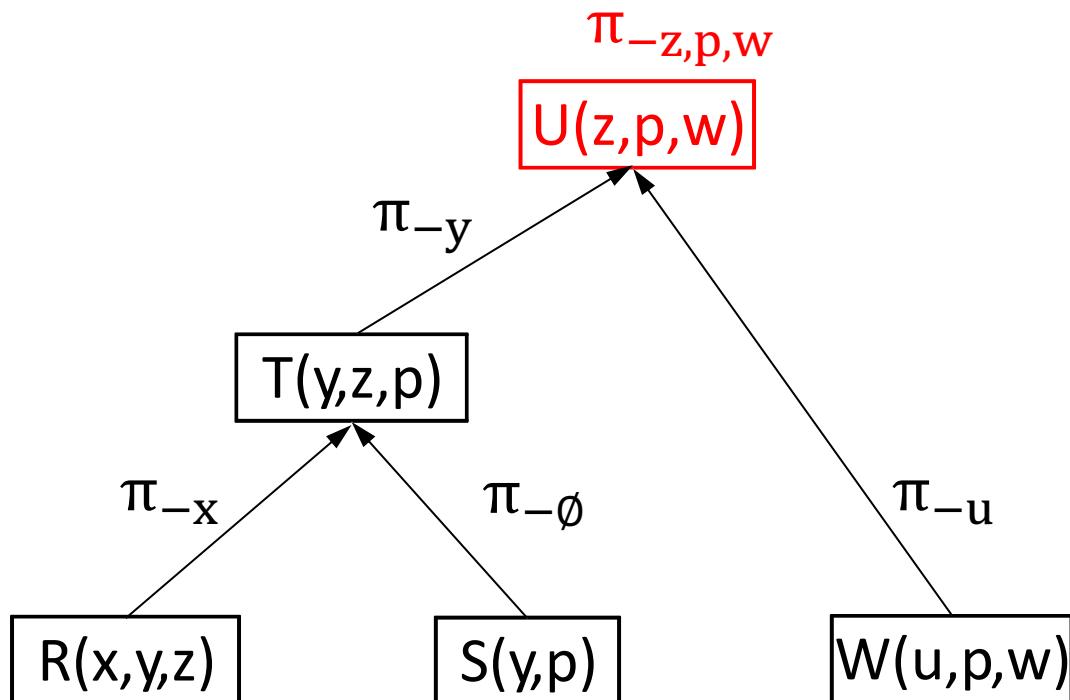
GYO reduction: Example 2



- GYO Ear removal

$$Q :- R(x,y,z), S(y,p), T(y,z,p), U(z,u,p), W(u,p,w).$$

- remove ears (= edges)
 - remove isolated nodes (variables)
 - remove consumed or empty edges (atoms)



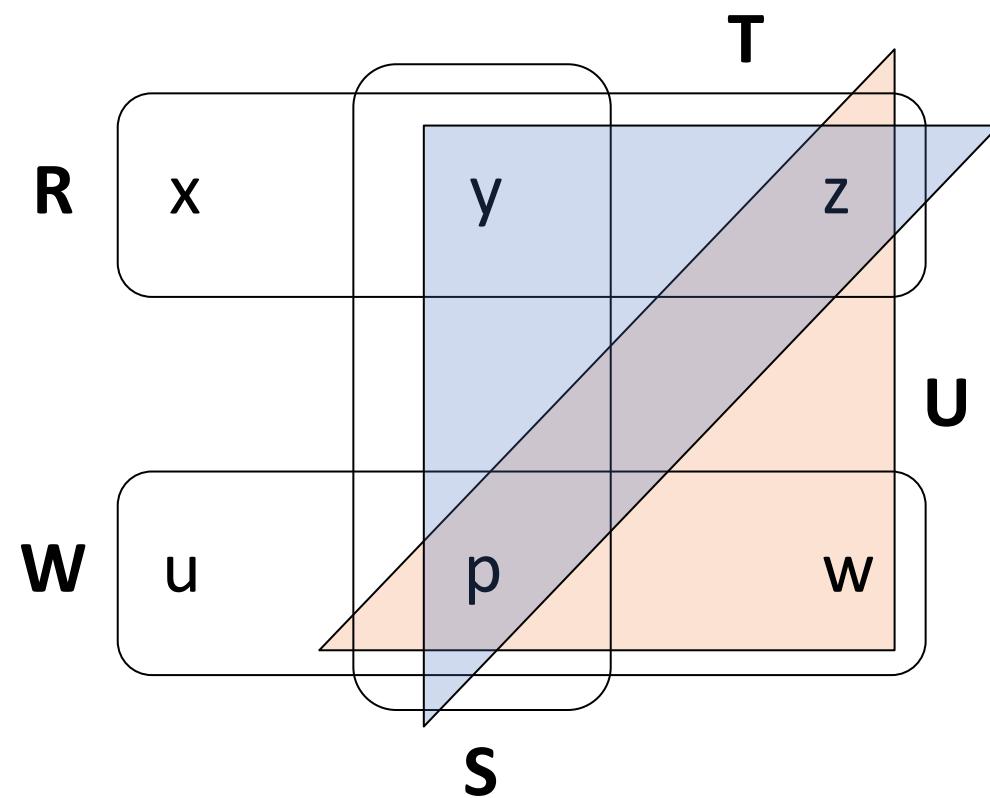
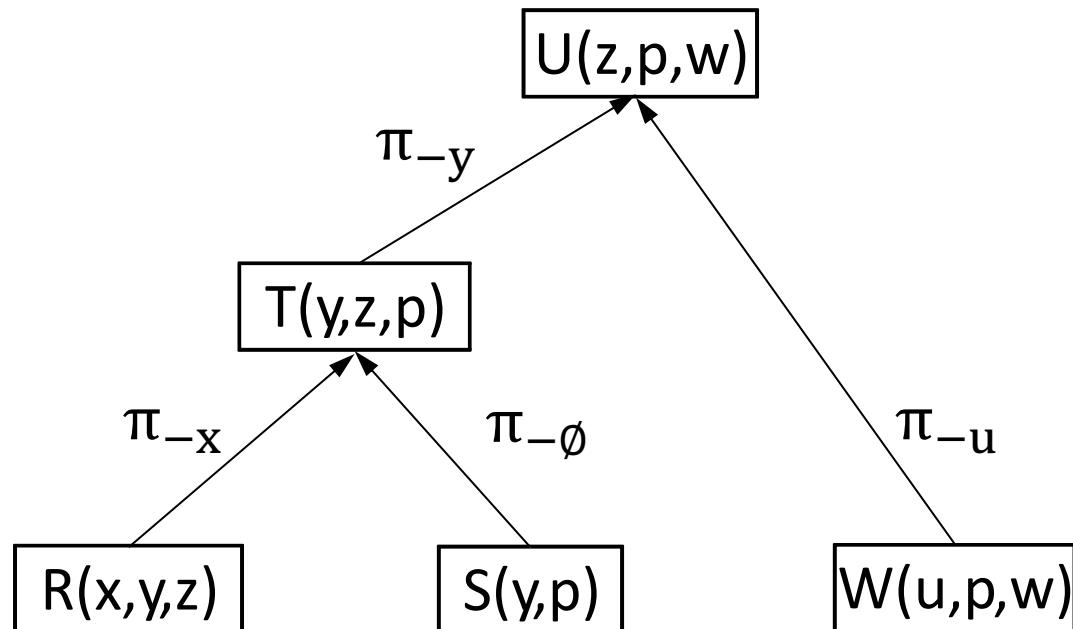
GYO reduction: Example 2



- GYO Ear removal

$Q :- R(x,y,z), S(y,p), T(y,z,p), U(z,u,p), W(u,p,w).$

- remove ears (= edges)
 - remove isolated nodes (variables)
 - remove consumed or empty edges (atoms)



GYO reduction: Example 2

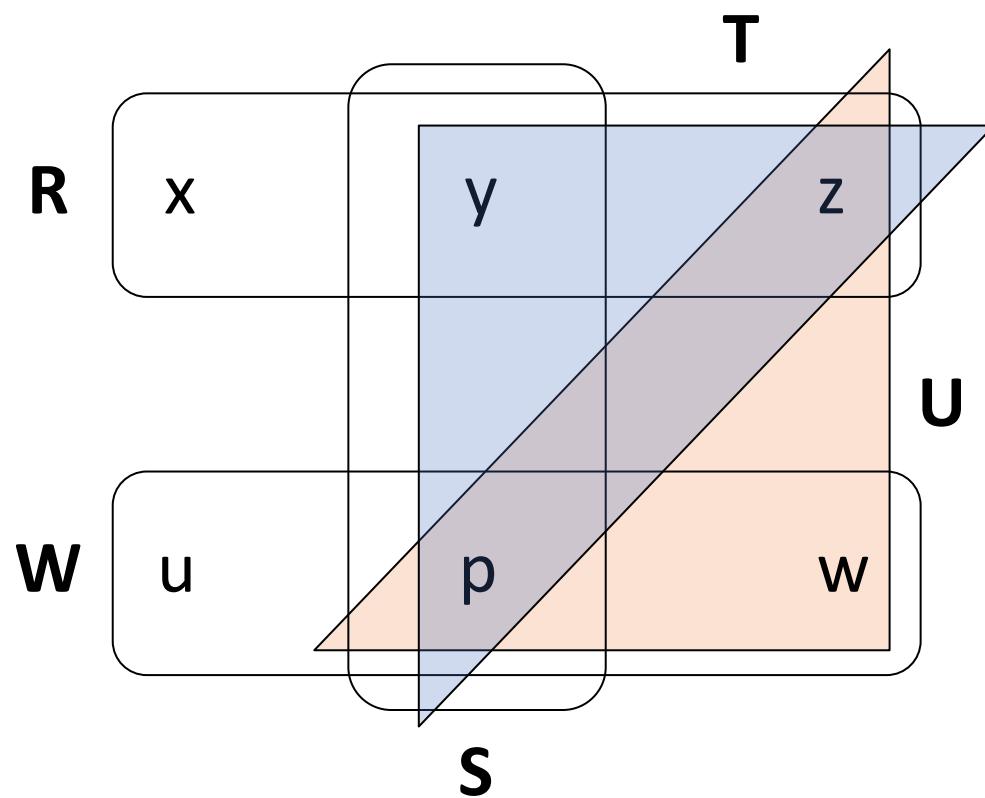
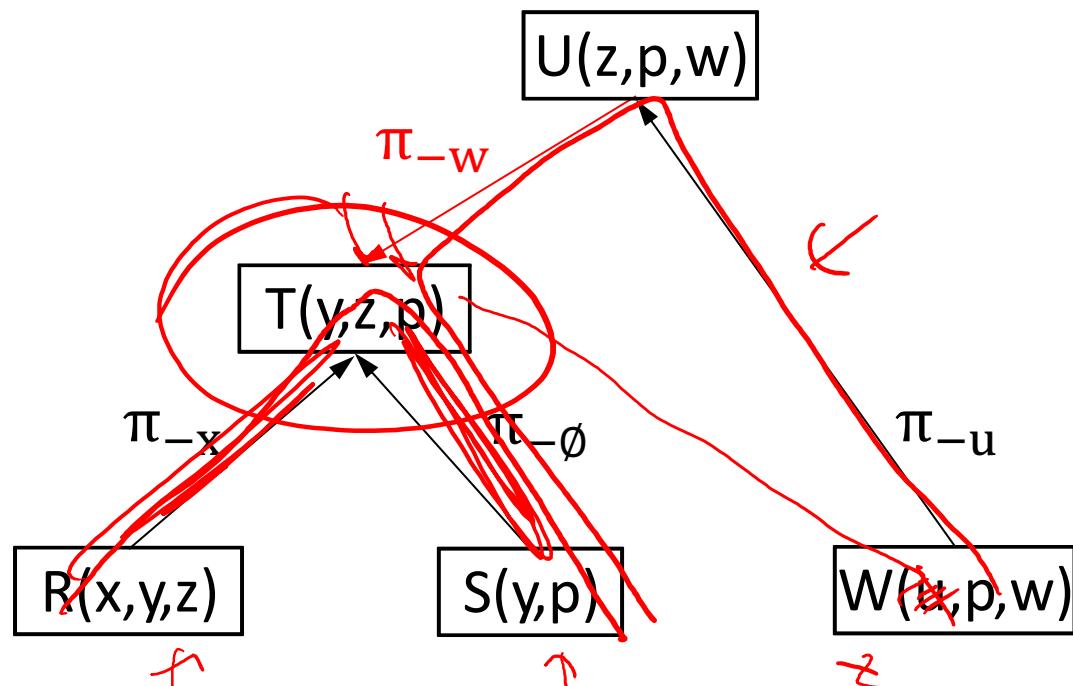


- GYO Ear removal

$$Q :- R(x,y,z), S(y,p), T(y,z,p), U(z,u,p), W(u,p,w).$$

- remove ears (= edges)

- remove isolated nodes (variables)
 - remove consumed or empty edges (atoms)





GYO reduction: Example 3: Triangle

- GYO Ear removal
 - remove ears (= edges)
 - remove isolated nodes (variables)
 - remove consumed or empty edges (atoms)

- What about the triangle query?

Join tree

?

Query hypergraph

?

GYO reduction: Example 3: Triangle

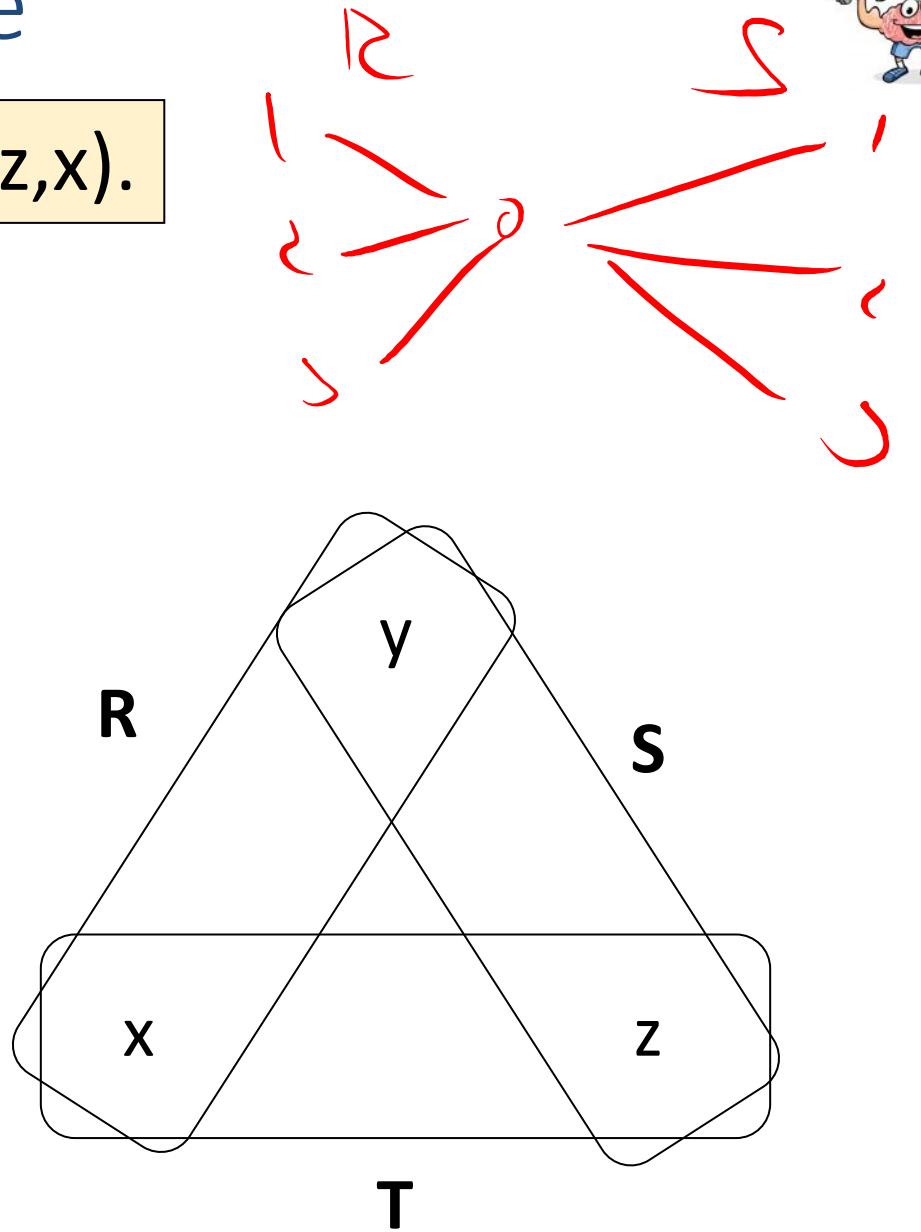
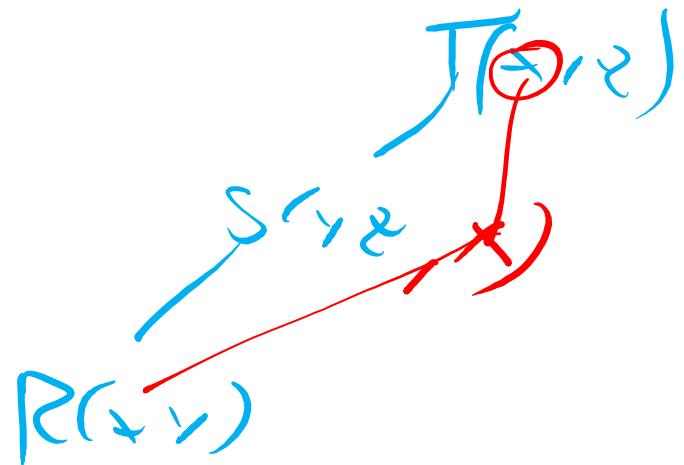


- GYO Ear removal

$Q :- R(x,y), S(y,z), T(z,x).$

- remove ears (= edges)
 - remove isolated nodes (variables)
 - remove consumed or empty edges (atoms)

- What about the triangle query?



GYO reduction: Example 4

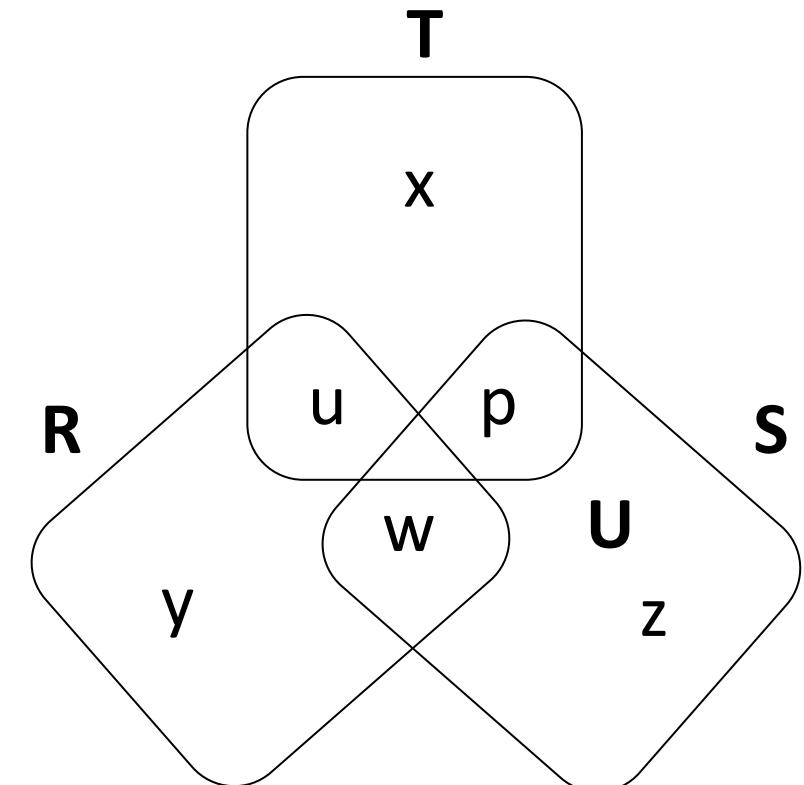
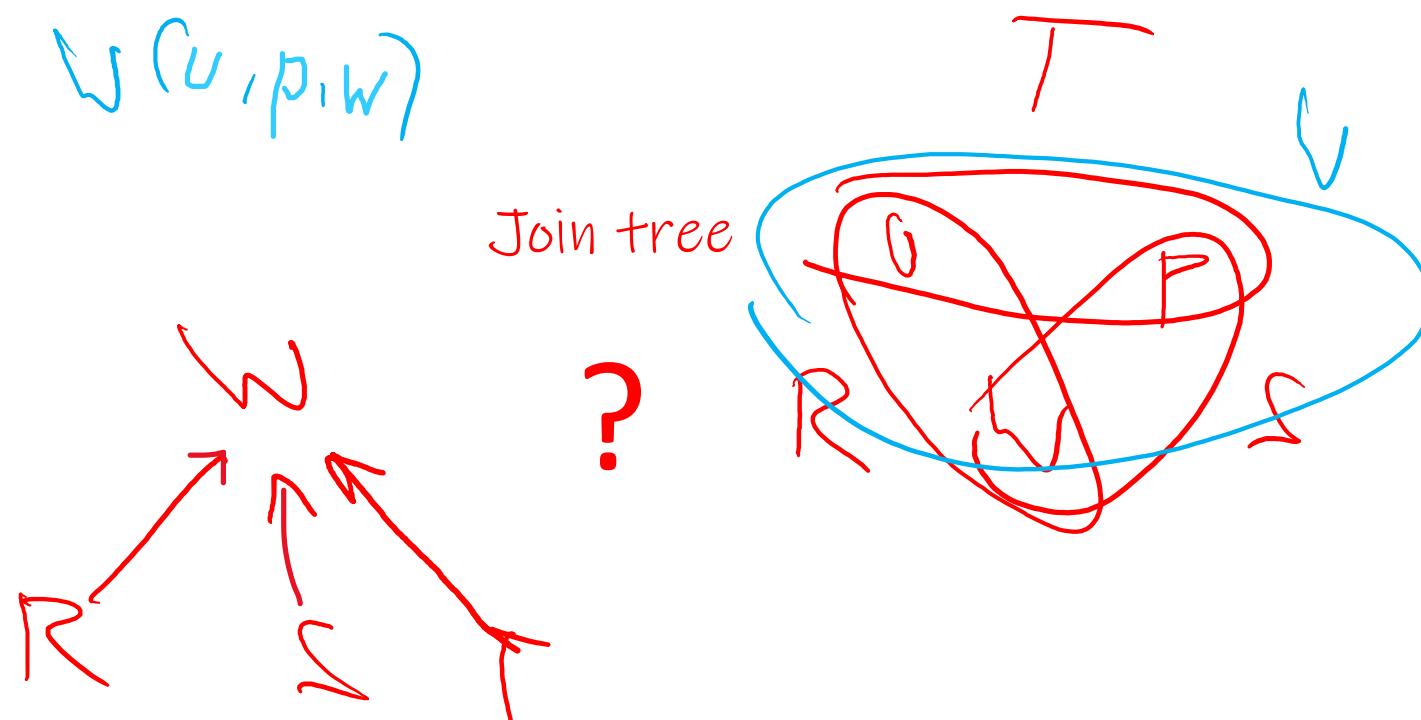


- GYO Ear removal

$Q :- R(y, u, w), S(z, p, w), T(x, u, p), \cancel{W(u, p, w)}.$

- remove ears (= edges)

- remove isolated nodes (variables)
- remove consumed or empty edges (atoms)



GYO reduction: Example 4



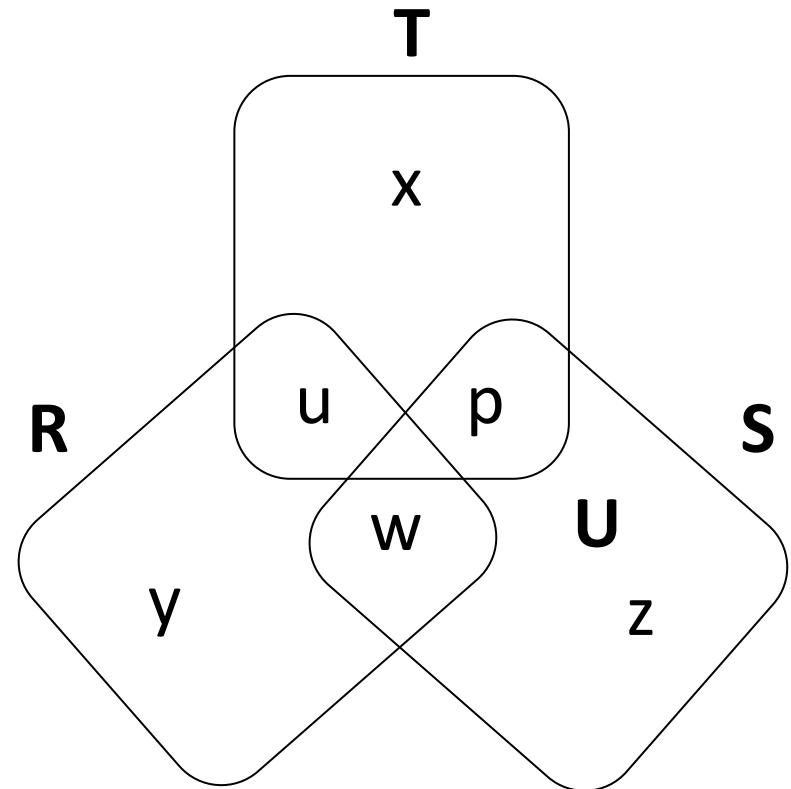
- GYO Ear removal

$$Q :- R(y,u,w), S(z,p,w), T(x,u,p), \cancel{W(u,p,w)}.$$

- remove ears (= edges)

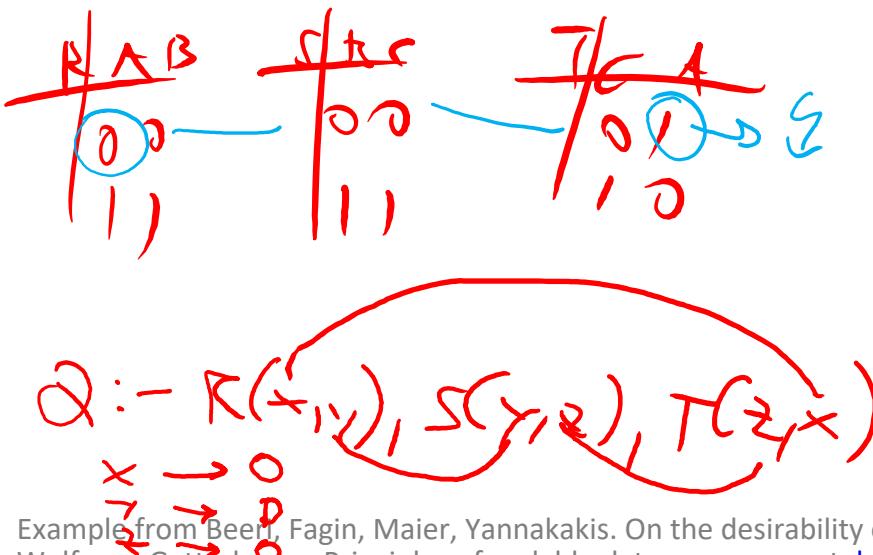
- remove isolated nodes (variables)
 - remove consumed or empty edges (atoms)

No Join tree, thus cyclic query!



Acyclic queries

- "Consistency" = no dangling tuples (recall semi-join)
 - locally join consistent $\pi_{R_i}(R_i \bowtie R_j) = R_i$
 - globally join consistent $\pi_{R_i}(R_1 \bowtie R_2 \bowtie \dots \bowtie R_k) = R_i$
- One implies the other
 - For all queries: global \Rightarrow local
 - Acyclic queries: local \Rightarrow global; but not for cyclic queries!



| | x | y | z |
|---------------------------|-----|-----|-----|
| R | 0 | 0 | 0 |
| S | 0 | 0 | 1 |
| T | 0 | 1 | 0 |
| $R \bowtie S$ | 0 | 0 | 0 |
| $(R \bowtie S) \bowtie T$ | 0 | 1 | 1 |
| Q | 1 | 1 | 1 |

Example: Consider $R(A,B) = \{(0,0), (1,1)\}$, $S(B,C) = \{(0,0), (1,1)\}$, $T(C,A) = \{(0,1), (1,0)\}$. Then:

- Any two relations are locally consistent.
- E.g. $R \bowtie S$ is $\{(0,0,0), (1,1,1)\}$, which projected onto R is $\{(0,0), (1,1)\}$
- But $R \bowtie S \bowtie T = \emptyset$, so the relations are not globally consistent

Reduction by Semi-joins

- Key insight for acyclic queries:
 - If there are no dangling tuples, then the result can never shrink with an additional join
 - Thus, for a "reduced database", every additional join can only increase the size of the intermediate query results

Repetition: Law of Semijoins

- Definition: the **semi-join** operation is:

$$R \ltimes_C S = \pi_{A_1, \dots, A_n}(R \bowtie S)$$

- Formally, $R \ltimes_C S$ means: retain from R only those tuples that have some matching tuple in S (C is a conjunction of join conditions)
- In bag semantics: duplicates in R are preserved / Duplicates in S don't matter
- Data complexity: $O(|R| + |S|)$ ignoring log-factors
- Input: $R(A_1, \dots, A_n)$, $S(B_1, \dots, B_m)$, Output: $T(A_1, \dots, A_n)$

- The **law of semijoins** is:

$$R \bowtie S = (R \ltimes_C S) \bowtie S$$

$$R(C \leftarrow Y \wedge Z), \quad S(X, Z, W)$$

- Thus, removing dangling tuples does not change the query result

Yannakakis Algorithm

- Given: acyclic full conjunctive query Q (full = no projections)
- Compute Q on any database in time $O(|\text{Input}| + |\text{Output}|)$ by using the **join tree**
- Step 1: **semi-join reduction** (two sweeps)
 - Pick any root node R in the join tree of Q
 - Step 1a: Do a semi-join reduction from the leaves to R (bottom-up)
 - Step 1b: Do a semi-join reduction from R to the leaves (top-down)
- Step 2: use the **join tree as query plan**: pick any root and join bottom-up or top-down
 - Notice that step 2 can be combined with the top-down SJ-reduction

Application

$Q(x,y,z,u,v,w) :- R(x,y), S(y,z), T(z,u), K(u,v), L(v,w).$

| R(x,y) | | S(y,z) | |
|----------------|----------------|----------------|----------------|
| x | y | y | z |
| a ₁ | b ₁ | b ₁ | c ₁ |
| a ₁ | b ₂ | b ₁ | c ₂ |
| a ₁ | b ₃ | b ₁ | c ₃ |
| a ₂ | b ₁ | b ₂ | c ₁ |
| a ₂ | b ₂ | b ₂ | c ₂ |
| a ₂ | b ₃ | b ₂ | c ₃ |
| a ₃ | b ₁ | b ₃ | c ₁ |
| a ₃ | b ₂ | b ₃ | c ₂ |
| a ₃ | b ₃ | b ₃ | c ₃ |

How many joins are between R and S?

?

Application

$Q(x,y,z,u,v,w) :- R(x,y), S(y,z), T(z,u), K(u,v), L(v,w).$

| R(x,y) | | S(y,z) | |
|----------------|----------------|----------------|----------------|
| x | y | y | z |
| a ₁ | b ₁ | b ₁ | c ₁ |
| a ₁ | b ₂ | b ₁ | c ₂ |
| a ₁ | b ₃ | b ₁ | c ₃ |
| a ₂ | b ₁ | b ₂ | c ₁ |
| a ₂ | b ₂ | b ₂ | c ₂ |
| a ₂ | b ₃ | b ₂ | c ₃ |
| a ₃ | b ₁ | b ₃ | c ₁ |
| a ₃ | b ₂ | b ₃ | c ₂ |
| a ₃ | b ₃ | b ₃ | c ₃ |

How many joins are between R and S?

27

How can we represent the joins
in a graph more compactly?

?

Application

$Q(x,y,z,u,v,w) :- R(x,y), S(y,z), T(z,u), K(u,v), L(v,w).$

| R(x,y) | | S(y,z) | |
|----------------|----------------|----------------|----------------|
| x | y | y | z |
| a ₁ | b ₁ | b ₁ | c ₁ |
| a ₁ | b ₂ | b ₁ | c ₂ |
| a ₁ | b ₃ | b ₁ | c ₃ |
| a ₂ | b ₁ | b ₂ | c ₁ |
| a ₂ | b ₂ | b ₂ | c ₂ |
| a ₂ | b ₃ | b ₂ | c ₃ |
| a ₃ | b ₁ | b ₃ | c ₁ |
| a ₃ | b ₂ | b ₃ | c ₂ |
| a ₃ | b ₃ | b ₃ | c ₃ |

How many joins are between R and S?

27

How can we represent the joins
in a graph more compactly?

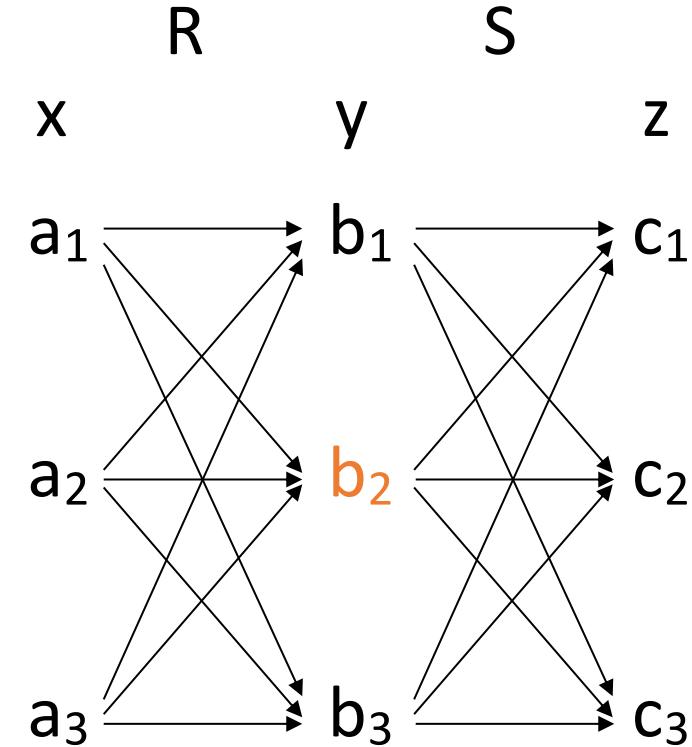
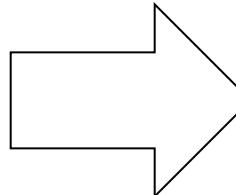
By looking using the dual (nodes as
domain values = joins) and counting paths!

Application

$Q(x,y,z,u,v,w) :- R(x,y), S(y,z), T(z,u), K(u,v), L(v,w).$

| R(x,y) | | S(y,z) | |
|----------------|----------------|----------------|----------------|
| x | y | y | z |
| a ₁ | b ₁ | b ₁ | c ₁ |
| a ₁ | b ₂ | b ₁ | c ₂ |
| a ₁ | b ₃ | b ₁ | c ₃ |
| a ₂ | b ₁ | b ₂ | c ₁ |
| a ₂ | b ₂ | b ₂ | c ₂ |
| a ₂ | b ₃ | b ₂ | c ₃ |
| a ₃ | b ₁ | b ₃ | c ₁ |
| a ₃ | b ₂ | b ₃ | c ₂ |
| a ₃ | b ₃ | b ₃ | c ₃ |

- Nodes are the tuples
- Edges are the joins



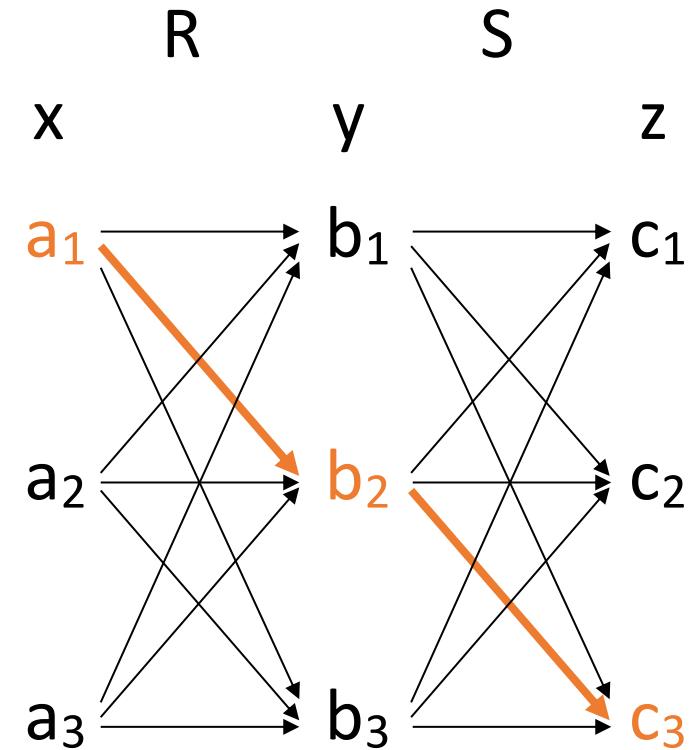
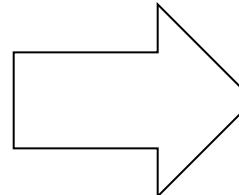
- Nodes are domain values
- Edges are the tuples
- Equi-joins b/w tuples as shared domain

Application

$Q(x,y,z,u,v,w) :- R(x,y), S(y,z), T(z,u), K(u,v), L(v,w).$

| R(x,y) | | S(y,z) | |
|----------------|----------------|----------------|----------------|
| x | y | y | z |
| a ₁ | b ₁ | b ₁ | c ₁ |
| a ₁ | b ₂ | b ₁ | c ₂ |
| a ₁ | b ₃ | b ₁ | c ₃ |
| a ₂ | b ₁ | b ₂ | c ₁ |
| a ₂ | b ₂ | b ₂ | c ₂ |
| a ₂ | b ₃ | b ₂ | c ₃ |
| a ₃ | b ₁ | b ₃ | c ₁ |
| a ₃ | b ₂ | b ₃ | c ₂ |
| a ₃ | b ₃ | b ₃ | c ₃ |

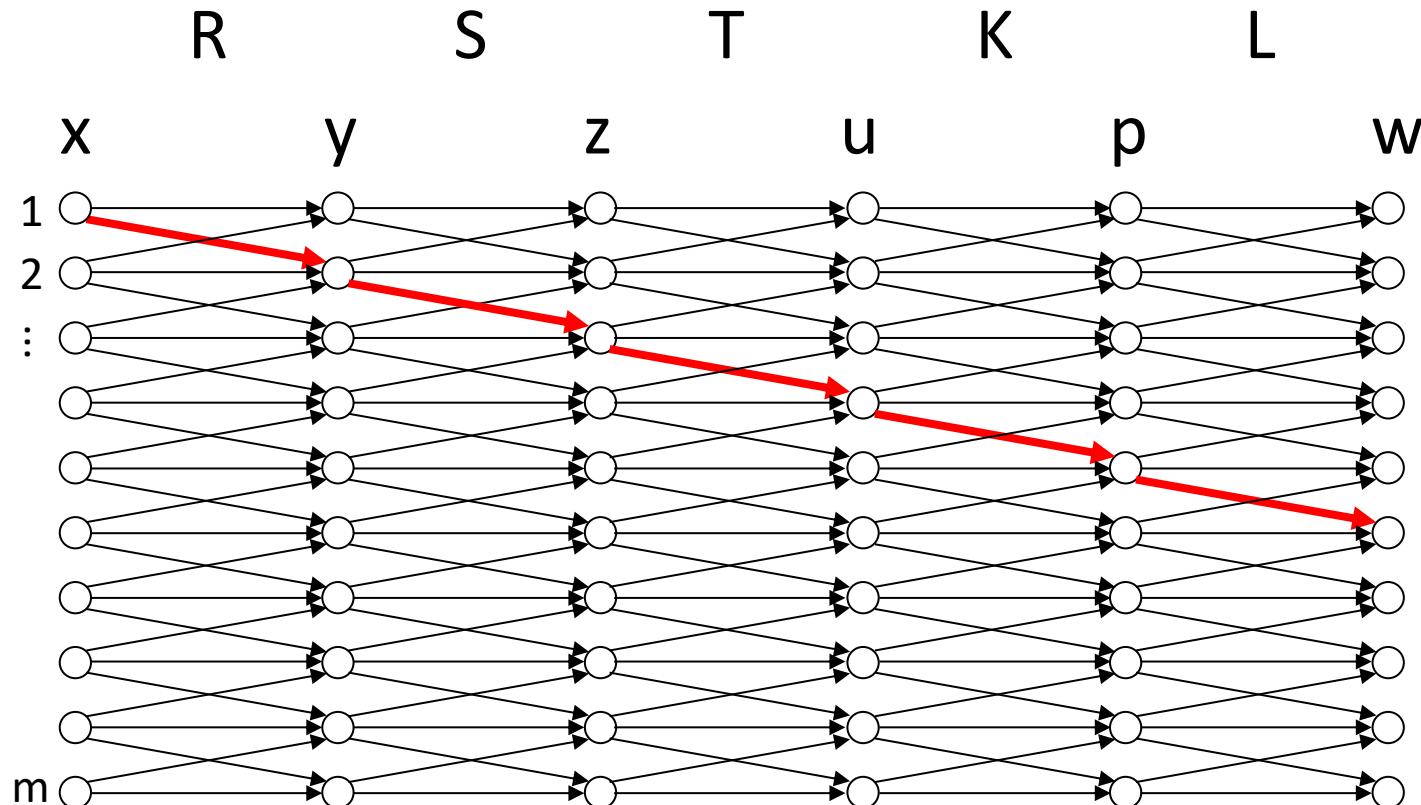
- Nodes are the tuples
- Edges are the joins



- Nodes are domain values
- Edges are the tuples
- Equi-joins b/w tuples as shared domain

Application

$Q(x,y,z,u,v,w) :- R(x,y), S(y,z), T(z,u), K(u,v), L(v,w).$



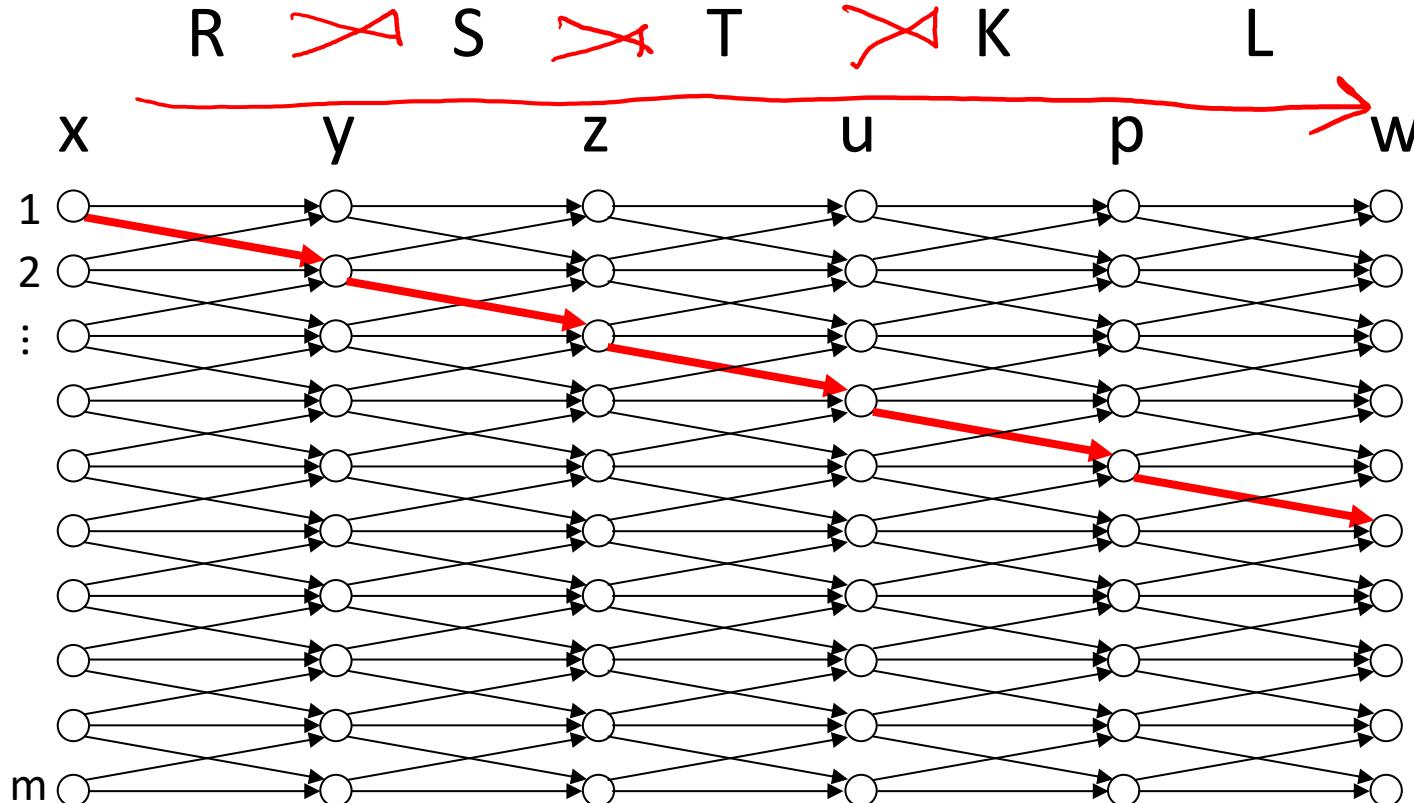
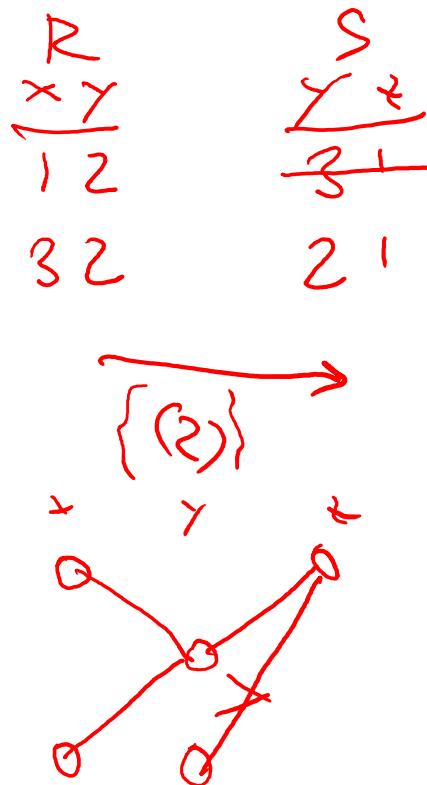
- Intermediate relations of size up to m^5 (m is here domain size!)
- But final answer can be as small as 0 (or 1...)

Application

$$\pi_x = \pi_{Lx}$$

$Q(x,y,z,u,v,w) :- R(x,y), S(y,z), T(z,u), K(u,v), L(v,w).$

Semi-join reduction allows pre-processing in $O(n)$ where n is size of DB (= number of edges)



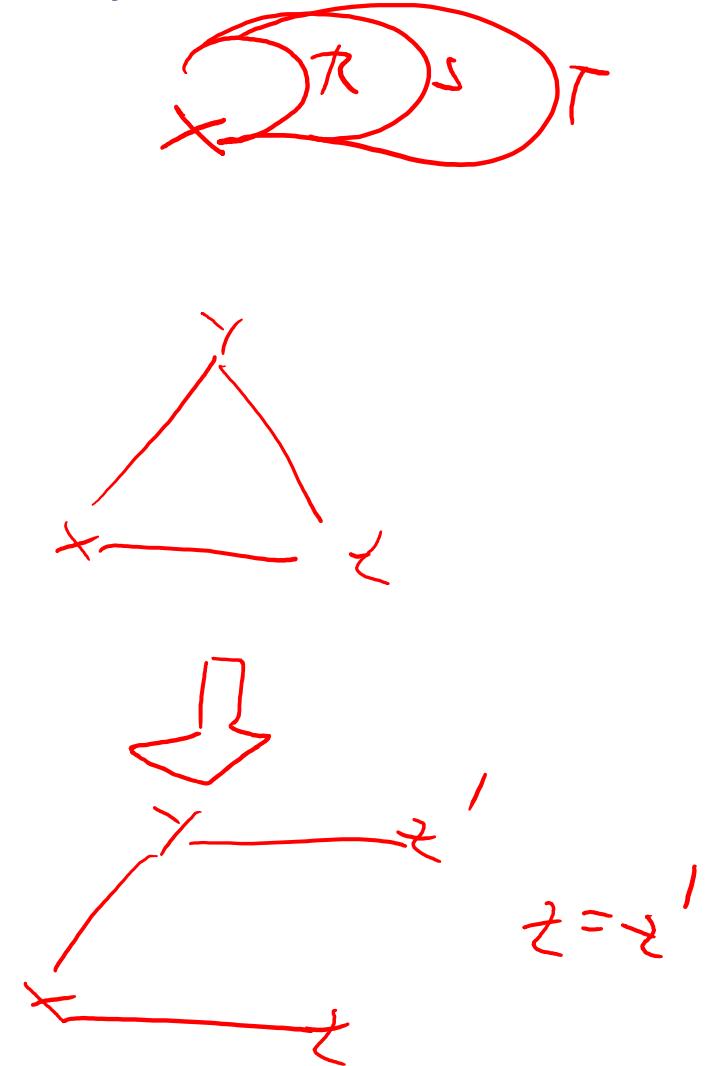
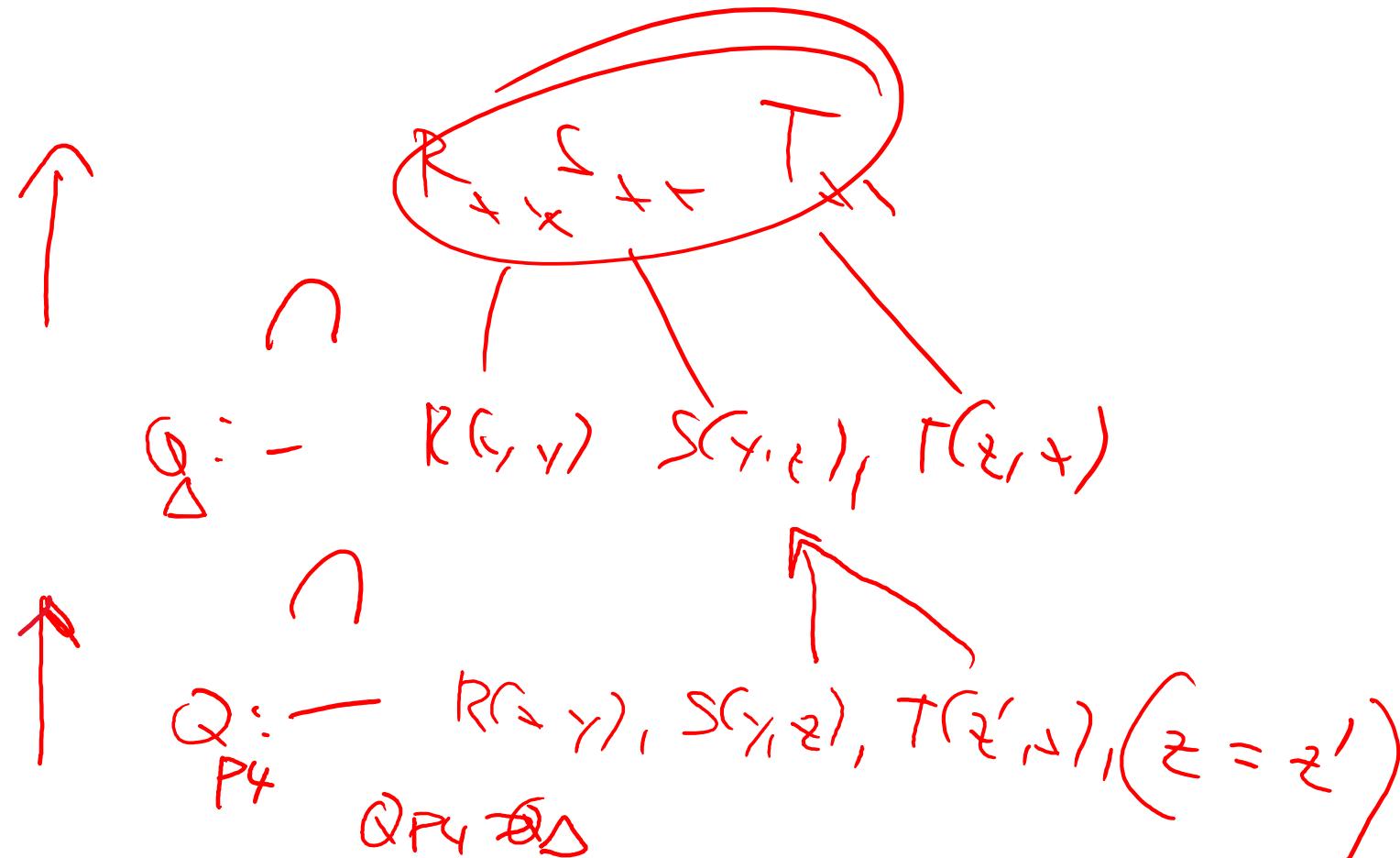
$S := S \bowtie R$
 $T := T \bowtie S$
 $K := K \bowtie T$
 $L := L \bowtie K$
 $K := K \bowtie L$
 $T := T \bowtie K$
 $S := S \bowtie T$
 $R := R \bowtie S$

FWD

BACK

- Intermediate relations of size up to m^5 (m is here domain size!)
- But final answer can be as small as 0 (or 1...)

Over- and under approximation via query containment



Outline: T3-1: Acyclic conjunctive queries

- T3-1: Acyclic conjunctive queries
 - The semijoin operator
 - Join trees & Yannakakis algorithm
 - Query hypergraphs & GYO reduction
 - A detailed Yannakakis example
 - Full semijoin reductions
- T3-2: Cyclic conjunctive queries

Yannakakis Algorithm example: start from join tree

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

Hypergraph

?

| R(y,z) | |
|----------------|----------------|
| y | z |
| b ₁ | c ₁ |
| b ₁ | c ₂ |
| b ₄ | c ₆ |

one of several
possible ones

Rooted Join tree

| S(p,w) | |
|----------------|----------------|
| p | w |
| e ₁ | f ₁ |
| e ₁ | f ₂ |
| e ₄ | f ₆ |

| T(x,y,z) | | |
|----------------|----------------|----------------|
| x | y | z |
| a ₁ | b ₁ | c ₁ |
| a ₁ | b ₁ | c ₂ |
| a ₃ | b ₃ | c ₁ |
| a ₃ | b ₁ | c ₄ |
| a ₂ | b ₂ | c ₃ |

?

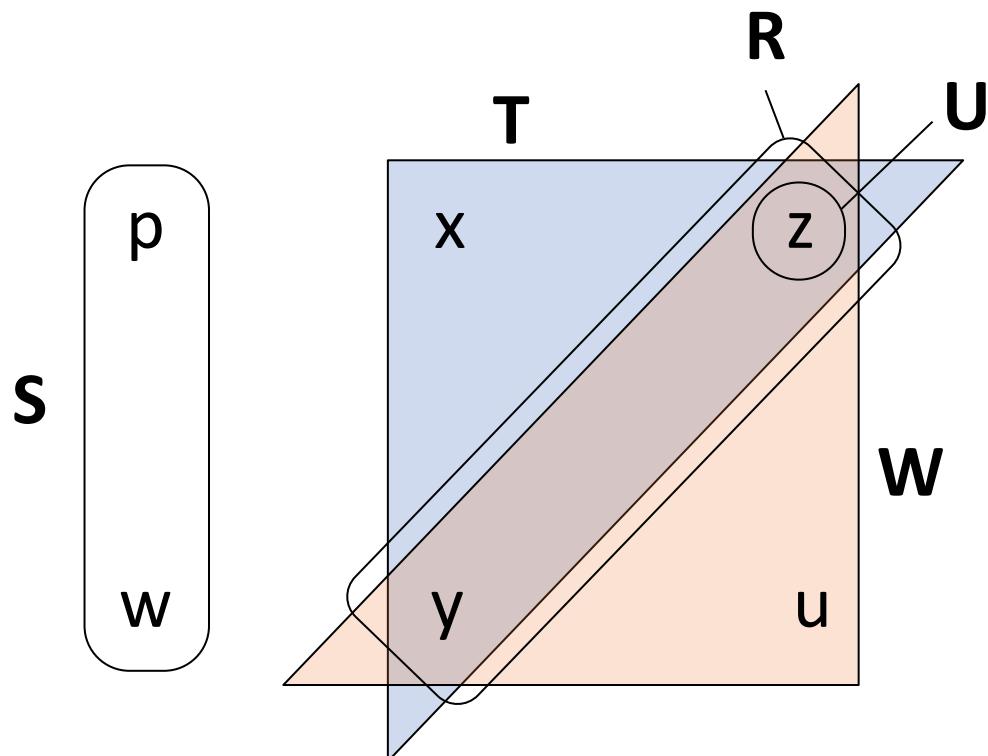
| U(z) | |
|----------------|--|
| z | |
| c ₁ | |
| c ₂ | |
| c ₃ | |

| W(y,z,u) | | |
|----------------|----------------|----------------|
| y | z | u |
| b ₁ | c ₁ | d ₁ |
| b ₁ | c ₂ | d ₁ |
| b ₁ | c ₂ | d ₂ |
| b ₂ | c ₂ | d ₂ |

Yannakakis Algorithm example: start from join tree

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

Hypergraph



| R(y,z) | |
|----------------|----------------|
| y | z |
| b ₁ | c ₁ |
| b ₁ | c ₂ |
| b ₄ | c ₆ |

Rooted Join tree

| S(p,w) | |
|----------------|----------------|
| p | w |
| e ₁ | f ₁ |
| e ₁ | f ₂ |
| e ₄ | f ₆ |

| T(x,y,z) | | |
|----------------|----------------|----------------|
| x | y | z |
| a ₁ | b ₁ | c ₁ |
| a ₁ | b ₁ | c ₂ |
| a ₃ | b ₃ | c ₁ |
| a ₃ | b ₁ | c ₄ |
| a ₂ | b ₂ | c ₃ |

?

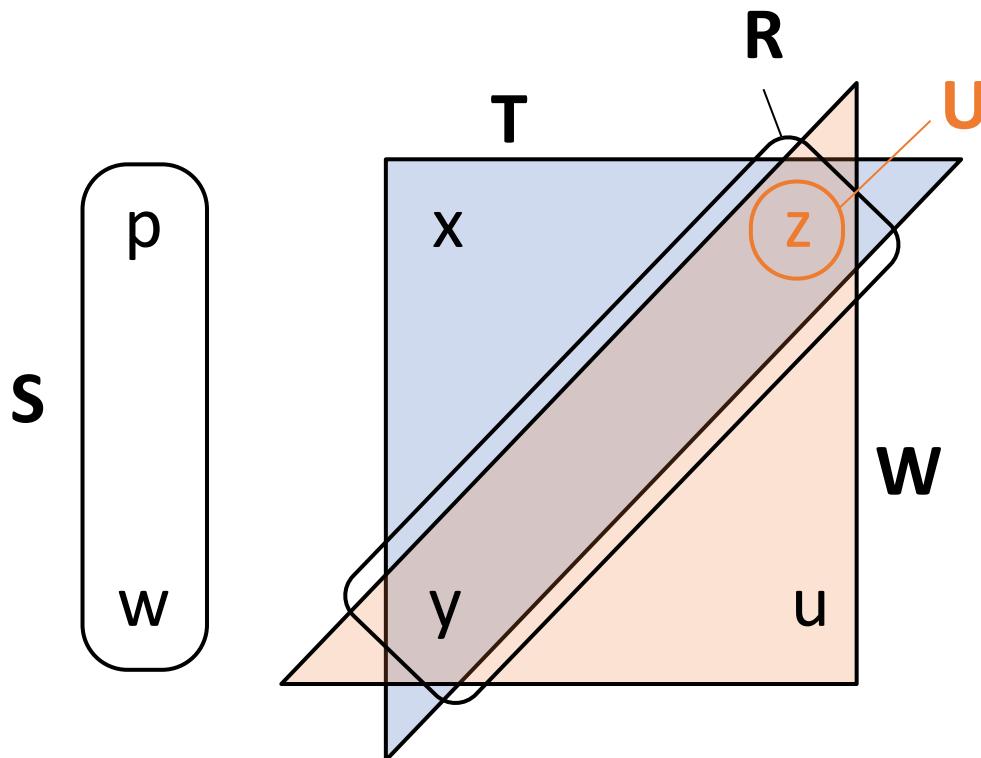
| U(z) | |
|----------------|--|
| z | |
| c ₁ | |
| c ₂ | |
| c ₃ | |

| W(y,z,u) | | |
|----------------|----------------|----------------|
| y | z | u |
| b ₁ | c ₁ | d ₁ |
| b ₁ | c ₂ | d ₁ |
| b ₁ | c ₂ | d ₂ |
| b ₂ | c ₂ | d ₂ |

Yannakakis Algorithm example: start from join tree

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

Hypergraph



| R(y,z) | |
|----------------|----------------|
| y | z |
| b ₁ | c ₁ |
| b ₁ | c ₂ |
| b ₄ | c ₆ |

Rooted Join tree

| S(p,w) | |
|----------------|----------------|
| p | w |
| e ₁ | f ₁ |
| e ₁ | f ₂ |
| e ₄ | f ₆ |

| T(x,y,z) | | |
|----------------|----------------|----------------|
| x | y | z |
| a ₁ | b ₁ | c ₁ |
| a ₁ | b ₁ | c ₂ |
| a ₃ | b ₃ | c ₁ |
| a ₃ | b ₁ | c ₄ |
| a ₂ | b ₂ | c ₃ |

$$z = -\emptyset$$

| U(z) | |
|----------------|--|
| z | |
| c ₁ | |
| c ₂ | |
| c ₃ | |

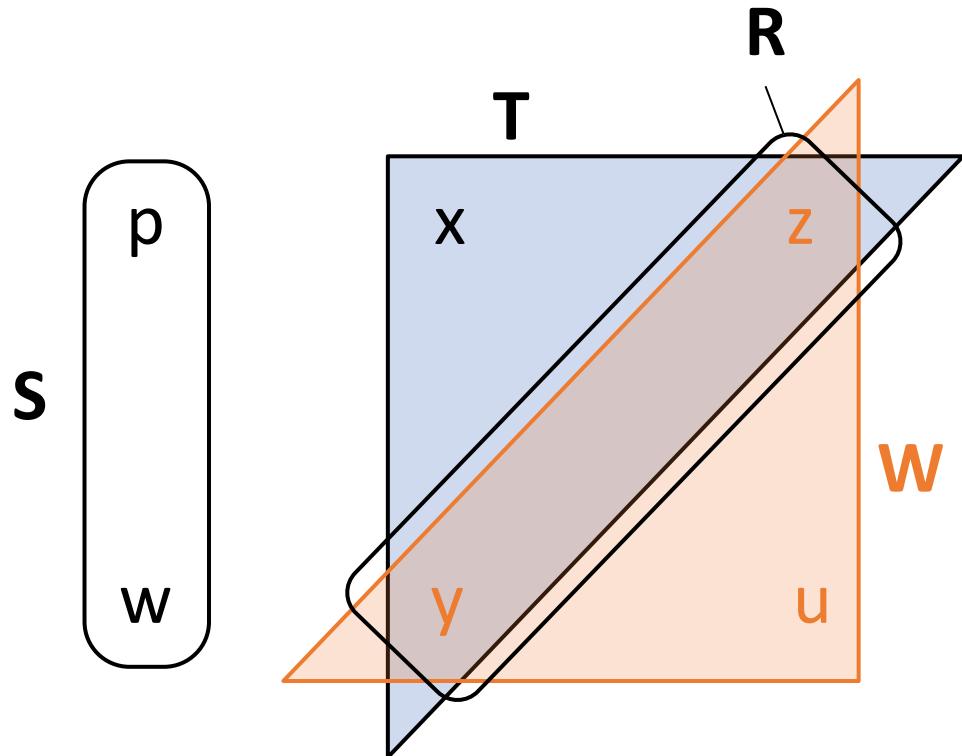
| W(y,z,u) | | |
|----------------|----------------|----------------|
| y | z | u |
| b ₁ | c ₁ | d ₁ |
| b ₁ | c ₂ | d ₁ |
| b ₁ | c ₂ | d ₂ |
| b ₂ | c ₂ | d ₂ |

Yannakakis Algorithm example: start from join tree

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

| R(y,z) | |
|----------------|----------------|
| y | z |
| b ₁ | c ₁ |
| b ₁ | c ₂ |
| b ₄ | c ₆ |

Hypergraph



Rooted Join tree

| S(p,w) | |
|----------------|----------------|
| p | w |
| e ₁ | f ₁ |
| e ₁ | f ₂ |
| e ₄ | f ₆ |

| T(x,y,z) | | |
|----------------|----------------|----------------|
| x | y | z |
| a ₁ | b ₁ | c ₁ |
| a ₁ | b ₁ | c ₂ |
| a ₃ | b ₃ | c ₁ |
| a ₃ | b ₁ | c ₄ |
| a ₂ | b ₂ | c ₃ |

shared variable

$z = -\emptyset$

$y, z = -u$

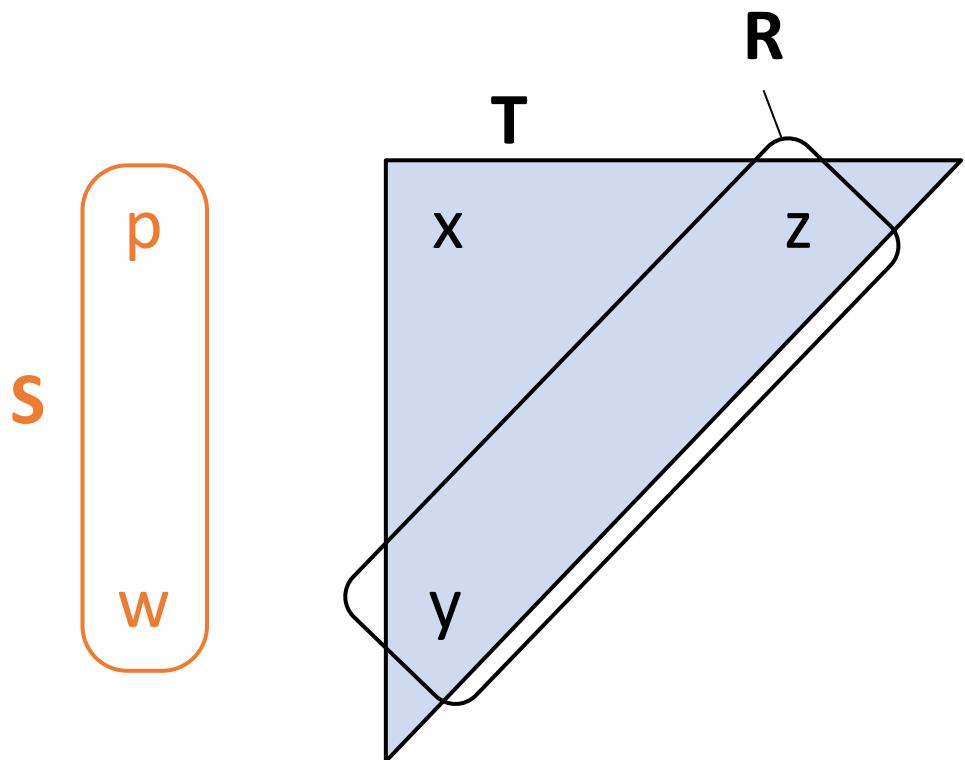
| U(z) | | |
|----------------|--|--|
| z | | |
| c ₁ | | |
| c ₂ | | |
| c ₃ | | |

| W(y,z,u) | | |
|----------------|----------------|----------------|
| y | z | u |
| b ₁ | c ₁ | d ₁ |
| b ₁ | c ₂ | d ₁ |
| b ₁ | c ₂ | d ₂ |
| b ₂ | c ₂ | d ₂ |

Yannakakis Algorithm example: start from join tree

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

Hypergraph



| R(y,z) | |
|----------------|----------------|
| y | z |
| b ₁ | c ₁ |
| b ₁ | c ₂ |
| b ₄ | c ₆ |

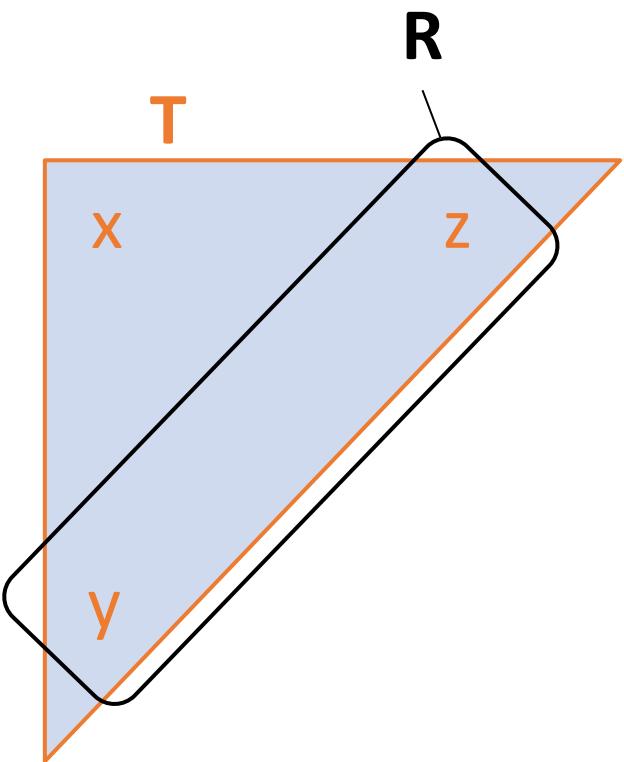
Rooted Join tree

| $\emptyset = -p, w$ | $S(p,w)$ | $T(x,y,z)$ | $U(z)$ | $W(y,z,u)$ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------------|--|----------------|-------------|----------------|----------------|----------------|----------------|----------------|----------------|--|---|---|---|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|--|---|----------------|----------------|----------------|---|---|---|---|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| | <table border="1"><thead><tr><th>p</th><th>w</th></tr></thead><tbody><tr><td>e₁</td><td>f₁</td></tr><tr><td>e₁</td><td>f₂</td></tr><tr><td>e₄</td><td>f₆</td></tr></tbody></table> | p | w | e ₁ | f ₁ | e ₁ | f ₂ | e ₄ | f ₆ | <table border="1"><thead><tr><th>x</th><th>y</th><th>z</th></tr></thead><tbody><tr><td>a₁</td><td>b₁</td><td>c₁</td></tr><tr><td>a₁</td><td>b₁</td><td>c₂</td></tr><tr><td>a₃</td><td>b₃</td><td>c₁</td></tr><tr><td>a₃</td><td>b₁</td><td>c₄</td></tr><tr><td>a₂</td><td>b₂</td><td>c₃</td></tr></tbody></table> | x | y | z | a ₁ | b ₁ | c ₁ | a ₁ | b ₁ | c ₂ | a ₃ | b ₃ | c ₁ | a ₃ | b ₁ | c ₄ | a ₂ | b ₂ | c ₃ | <table border="1"><thead><tr><th>z</th></tr></thead><tbody><tr><td>c₁</td></tr><tr><td>c₂</td></tr><tr><td>c₃</td></tr></tbody></table> | z | c ₁ | c ₂ | c ₃ | <table border="1"><thead><tr><th>y</th><th>z</th><th>u</th></tr></thead><tbody><tr><td>b₁</td><td>c₁</td><td>d₁</td></tr><tr><td>b₁</td><td>c₂</td><td>d₁</td></tr><tr><td>b₁</td><td>c₂</td><td>d₂</td></tr><tr><td>b₂</td><td>c₂</td><td>d₂</td></tr></tbody></table> | y | z | u | b ₁ | c ₁ | d ₁ | b ₁ | c ₂ | d ₁ | b ₁ | c ₂ | d ₂ | b ₂ | c ₂ | d ₂ |
| p | w | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| e ₁ | f ₁ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| e ₁ | f ₂ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| e ₄ | f ₆ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| x | y | z | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| a ₁ | b ₁ | c ₁ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| a ₁ | b ₁ | c ₂ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| a ₃ | b ₃ | c ₁ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| a ₃ | b ₁ | c ₄ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| a ₂ | b ₂ | c ₃ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| z | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| c ₁ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| c ₂ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| c ₃ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| y | z | u | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| b ₁ | c ₁ | d ₁ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| b ₁ | c ₂ | d ₁ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| b ₁ | c ₂ | d ₂ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| b ₂ | c ₂ | d ₂ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| $z = -\emptyset$ | | | $y, z = -u$ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Yannakakis Algorithm example: start from join tree

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

Hypergraph



| R(y,z) | |
|----------------|----------------|
| y | z |
| b ₁ | c ₁ |
| b ₁ | c ₂ |
| b ₄ | c ₆ |

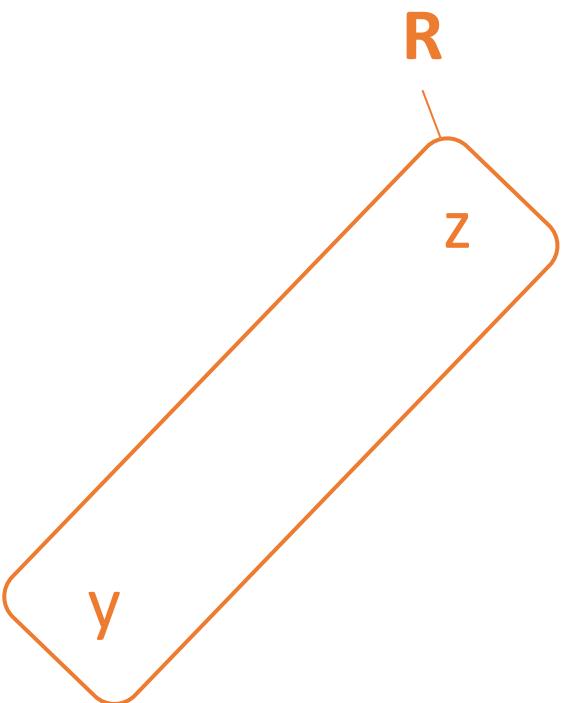
Rooted Join tree

| $\emptyset = -p, w$ | $y, z = -x$ | | | |
|---------------------|----------------|----------------|----------------|----------------|
| $S(p,w)$ | $T(x,y,z)$ | | | |
| p | w | x | y | z |
| e ₁ | f ₁ | a ₁ | b ₁ | c ₁ |
| e ₁ | f ₂ | a ₁ | b ₁ | c ₂ |
| e ₄ | f ₆ | a ₃ | b ₃ | c ₁ |
| | | a ₃ | b ₁ | c ₄ |
| | | a ₂ | b ₂ | c ₃ |
| $z = -\emptyset$ | $y, z = -u$ | | | |
| $U(z)$ | $W(y,z,u)$ | | | |
| z | y | z | u | |
| c ₁ | b ₁ | c ₁ | d ₁ | |
| c ₂ | b ₁ | c ₂ | d ₁ | |
| c ₃ | b ₁ | c ₂ | d ₂ | |
| | b ₂ | c ₂ | d ₂ | |

Yannakakis Algorithm example: start from join tree

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

Hypergraph



| R(y,z) | |
|----------------|----------------|
| y | z |
| b ₁ | c ₁ |
| b ₁ | c ₂ |
| b ₄ | c ₆ |

Rooted Join tree

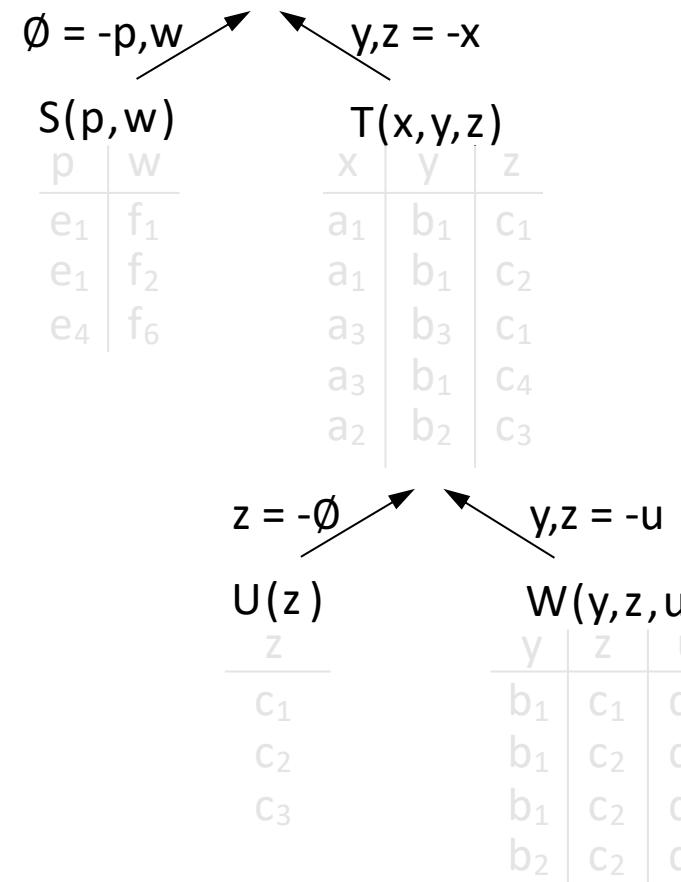
| $\emptyset = -p, w$ | $y, z = -x$ | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|----------------|----------------|----------------|----------------|---|----------------|----------------|----------------|--|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| $S(p, w)$ | $T(x, y, z)$ | | | | | | | | | | | | | | | | | | | | | | | | | | |
| <table border="1"><thead><tr><th>p</th><th>w</th></tr></thead><tbody><tr><td>e₁</td><td>f₁</td></tr><tr><td>e₁</td><td>f₂</td></tr><tr><td>e₄</td><td>f₆</td></tr></tbody></table> | p | w | e ₁ | f ₁ | e ₁ | f ₂ | e ₄ | f ₆ | <table border="1"><thead><tr><th>x</th><th>y</th><th>z</th></tr></thead><tbody><tr><td>a₁</td><td>b₁</td><td>c₁</td></tr><tr><td>a₁</td><td>b₁</td><td>c₂</td></tr><tr><td>a₃</td><td>b₃</td><td>c₁</td></tr><tr><td>a₃</td><td>b₁</td><td>c₄</td></tr><tr><td>a₂</td><td>b₂</td><td>c₃</td></tr></tbody></table> | x | y | z | a ₁ | b ₁ | c ₁ | a ₁ | b ₁ | c ₂ | a ₃ | b ₃ | c ₁ | a ₃ | b ₁ | c ₄ | a ₂ | b ₂ | c ₃ |
| p | w | | | | | | | | | | | | | | | | | | | | | | | | | | |
| e ₁ | f ₁ | | | | | | | | | | | | | | | | | | | | | | | | | | |
| e ₁ | f ₂ | | | | | | | | | | | | | | | | | | | | | | | | | | |
| e ₄ | f ₆ | | | | | | | | | | | | | | | | | | | | | | | | | | |
| x | y | z | | | | | | | | | | | | | | | | | | | | | | | | | |
| a ₁ | b ₁ | c ₁ | | | | | | | | | | | | | | | | | | | | | | | | | |
| a ₁ | b ₁ | c ₂ | | | | | | | | | | | | | | | | | | | | | | | | | |
| a ₃ | b ₃ | c ₁ | | | | | | | | | | | | | | | | | | | | | | | | | |
| a ₃ | b ₁ | c ₄ | | | | | | | | | | | | | | | | | | | | | | | | | |
| a ₂ | b ₂ | c ₃ | | | | | | | | | | | | | | | | | | | | | | | | | |
| $z = -\emptyset$ | $y, z = -u$ | | | | | | | | | | | | | | | | | | | | | | | | | | |
| $U(z)$ | $W(y, z, u)$ | | | | | | | | | | | | | | | | | | | | | | | | | | |
| <table border="1"><thead><tr><th>z</th></tr></thead><tbody><tr><td>c₁</td></tr><tr><td>c₂</td></tr><tr><td>c₃</td></tr></tbody></table> | z | c ₁ | c ₂ | c ₃ | <table border="1"><thead><tr><th>y</th><th>z</th><th>u</th></tr></thead><tbody><tr><td>b₁</td><td>c₁</td><td>d₁</td></tr><tr><td>b₁</td><td>c₂</td><td>d₁</td></tr><tr><td>b₁</td><td>c₂</td><td>d₂</td></tr><tr><td>b₂</td><td>c₂</td><td>d₂</td></tr></tbody></table> | y | z | u | b ₁ | c ₁ | d ₁ | b ₁ | c ₂ | d ₁ | b ₁ | c ₂ | d ₂ | b ₂ | c ₂ | d ₂ | | | | | | | |
| z | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| c ₁ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| c ₂ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| c ₃ | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| y | z | u | | | | | | | | | | | | | | | | | | | | | | | | | |
| b ₁ | c ₁ | d ₁ | | | | | | | | | | | | | | | | | | | | | | | | | |
| b ₁ | c ₂ | d ₁ | | | | | | | | | | | | | | | | | | | | | | | | | |
| b ₁ | c ₂ | d ₂ | | | | | | | | | | | | | | | | | | | | | | | | | |
| b ₂ | c ₂ | d ₂ | | | | | | | | | | | | | | | | | | | | | | | | | |

Yannakakis Algorithm example: start from join tree

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

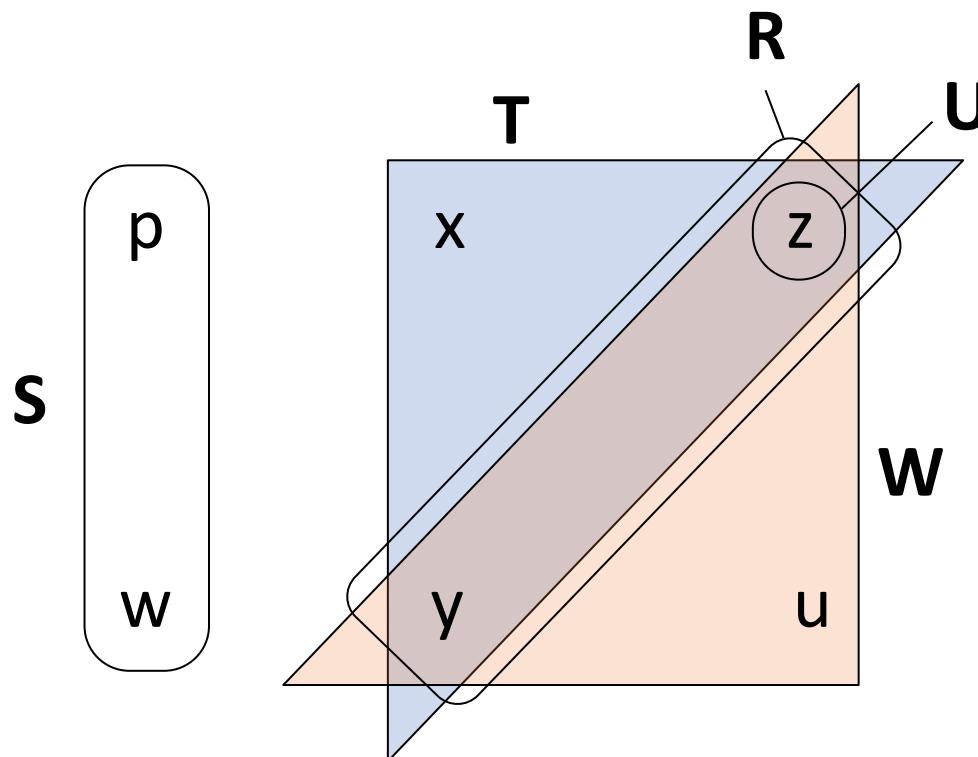
| R(y,z) | |
|----------------|----------------|
| y | z |
| b ₁ | c ₁ |
| b ₁ | c ₂ |
| b ₄ | c ₆ |

Rooted Join tree



Yannakakis Algorithm example: start from join tree

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$



| $R(y,z)$ | |
|----------|-------|
| y | z |
| b_1 | c_1 |
| b_1 | c_2 |
| b_4 | c_6 |

Rooted Join tree

$\emptyset = -p, w$

$S(p,w)$

| p | w |
|-------|-------|
| e_1 | f_1 |
| e_1 | f_2 |
| e_4 | f_6 |

$T(x,y,z)$

| x | y | z |
|-------|-------|-------|
| a_1 | b_1 | c_1 |
| a_1 | b_1 | c_2 |
| a_3 | b_3 | c_1 |
| a_3 | b_1 | c_4 |
| a_2 | b_2 | c_3 |

shared
variables

$z = -\emptyset$

$U(z)$

| z |
|-------|
| c_1 |
| c_2 |
| c_3 |

$W(y,z,u)$

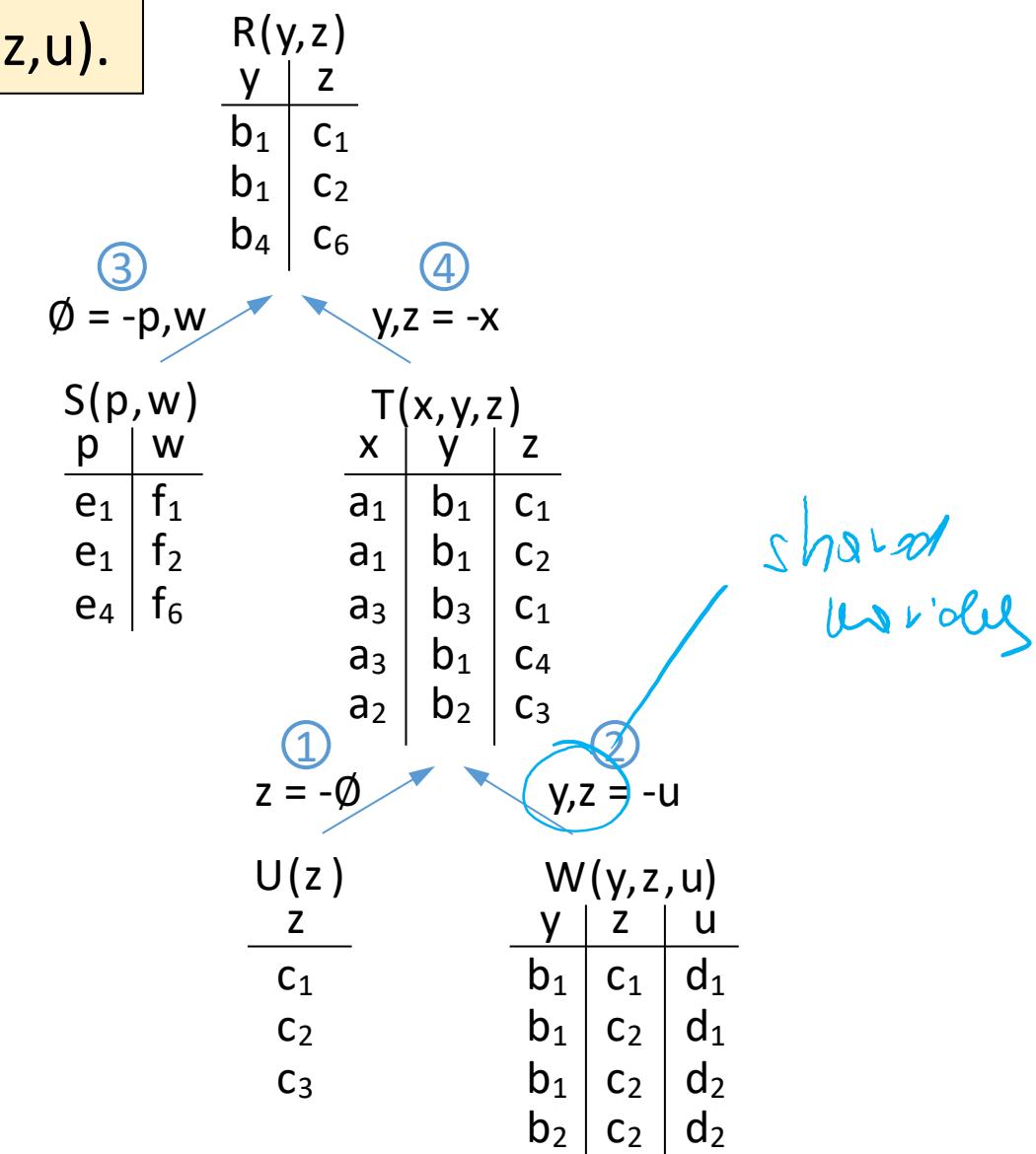
| y | z | u |
|-------|-------|-------|
| b_1 | c_1 | d_1 |
| b_1 | c_2 | d_1 |
| b_1 | c_2 | d_2 |
| b_2 | c_2 | d_2 |

Yannakakis Algorithm example: 1st pass

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

Semi-join phase \bowtie (remove dangling tuples) in $O(IN)$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order

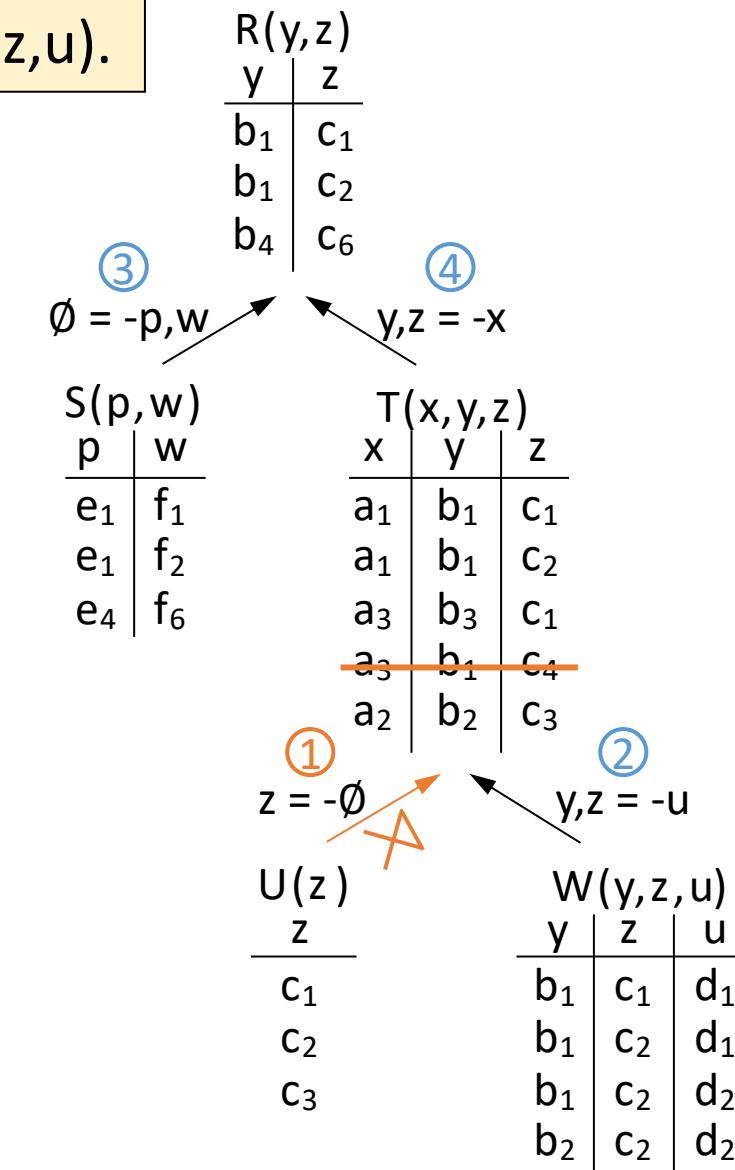


Yannakakis Algorithm example: 1st pass

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

Semi-join phase ✕ (remove dangling tuples) in $O(IN)$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order

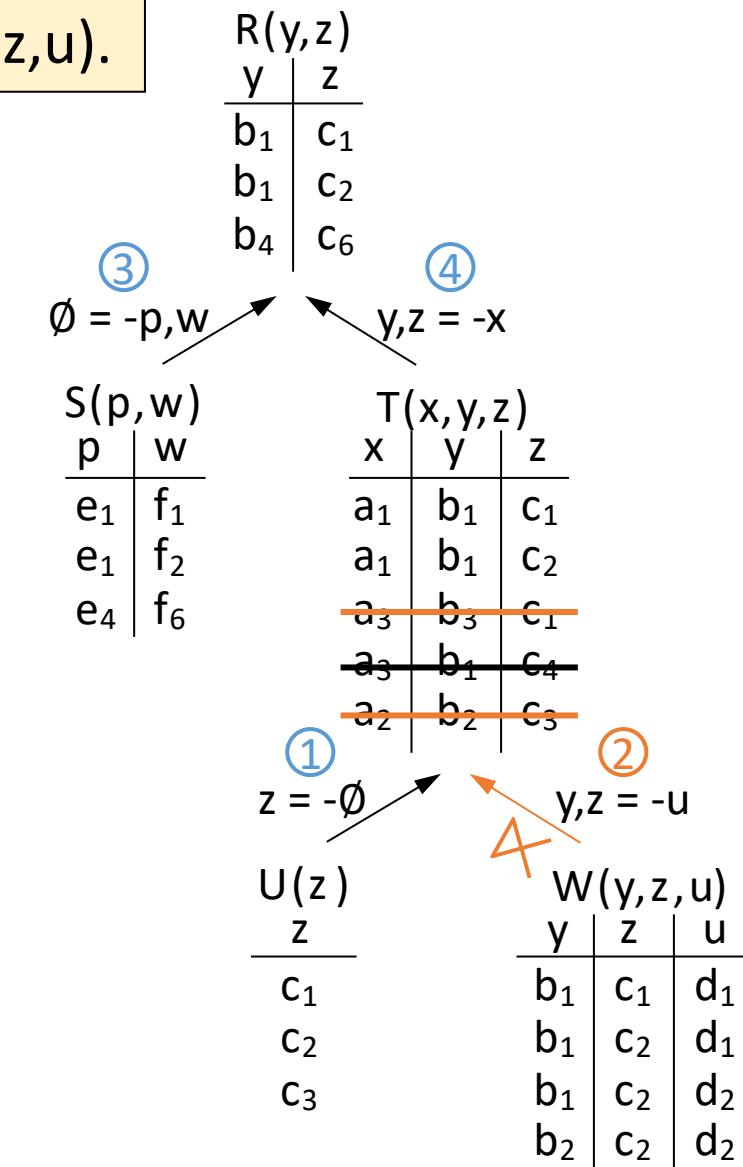


Yannakakis Algorithm example: 1st pass

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

Semi-join phase ✕ (remove dangling tuples) in $O(IN)$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order

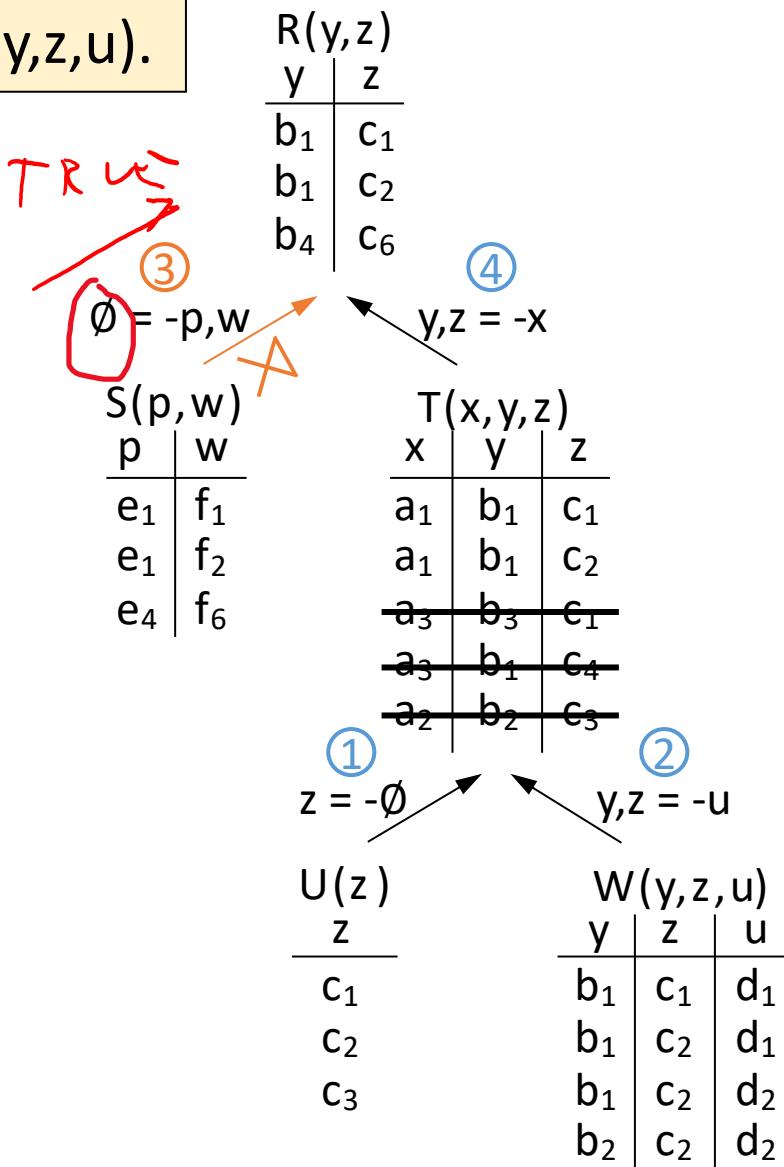


Yannakakis Algorithm example: 1st pass

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

Semi-join phase ✕ (remove dangling tuples) in $O(IN)$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order



Yannakakis Algorithm example: 1st pass

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

Semi-join phase ✕ (remove dangling tuples) in $O(IN)$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order

| R(y,z) | |
|----------------|----------------|
| y | z |
| b ₁ | c ₁ |
| b ₁ | c ₂ |
| b ₄ | c ₆ |

| S(p,w) | |
|----------------|----------------|
| p | w |
| e ₁ | f ₁ |
| e ₁ | f ₂ |
| e ₄ | f ₆ |

| T(x,y,z) | | |
|----------------|----------------|----------------|
| x | y | z |
| a ₁ | b ₁ | c ₁ |
| a ₁ | b ₁ | c ₂ |
| a ₃ | b ₃ | c ₁ |
| a ₃ | b ₁ | c ₄ |
| a ₂ | b ₂ | c ₃ |

| U(z) | |
|----------------|--|
| z | |
| c ₁ | |
| c ₂ | |
| c ₃ | |

| W(y,z,u) | | |
|----------------|----------------|----------------|
| y | z | u |
| b ₁ | c ₁ | d ₁ |
| b ₁ | c ₂ | d ₁ |
| b ₁ | c ₂ | d ₂ |
| b ₂ | c ₂ | d ₂ |

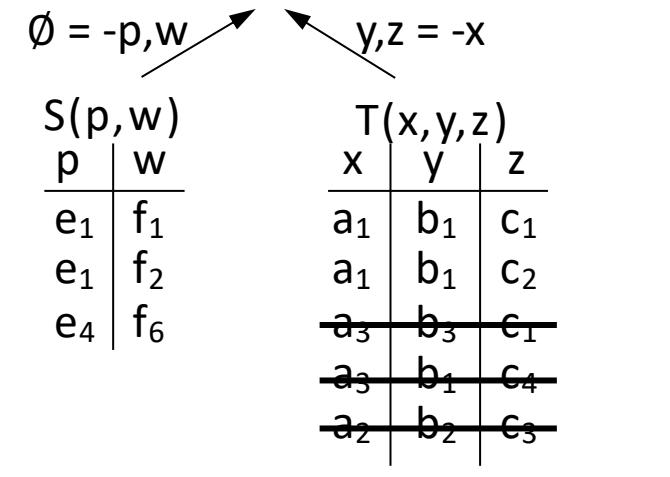
Yannakakis Algorithm example: 1st pass

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

Semi-join phase ✕ (remove dangling tuples) in $O(IN)$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order

| R(y,z) | |
|--------------------------|--------------------------|
| y | z |
| b ₁ | c ₁ |
| b ₁ | c ₂ |
| b₄ | c₆ |



Notice that at the end of the first pass, the table R at the root does not contain any more dangling tuples; it is completely reduced.

In other words, with a sequence of only local updates, we have aggregated at the root all necessary information to answer the Boolean query.

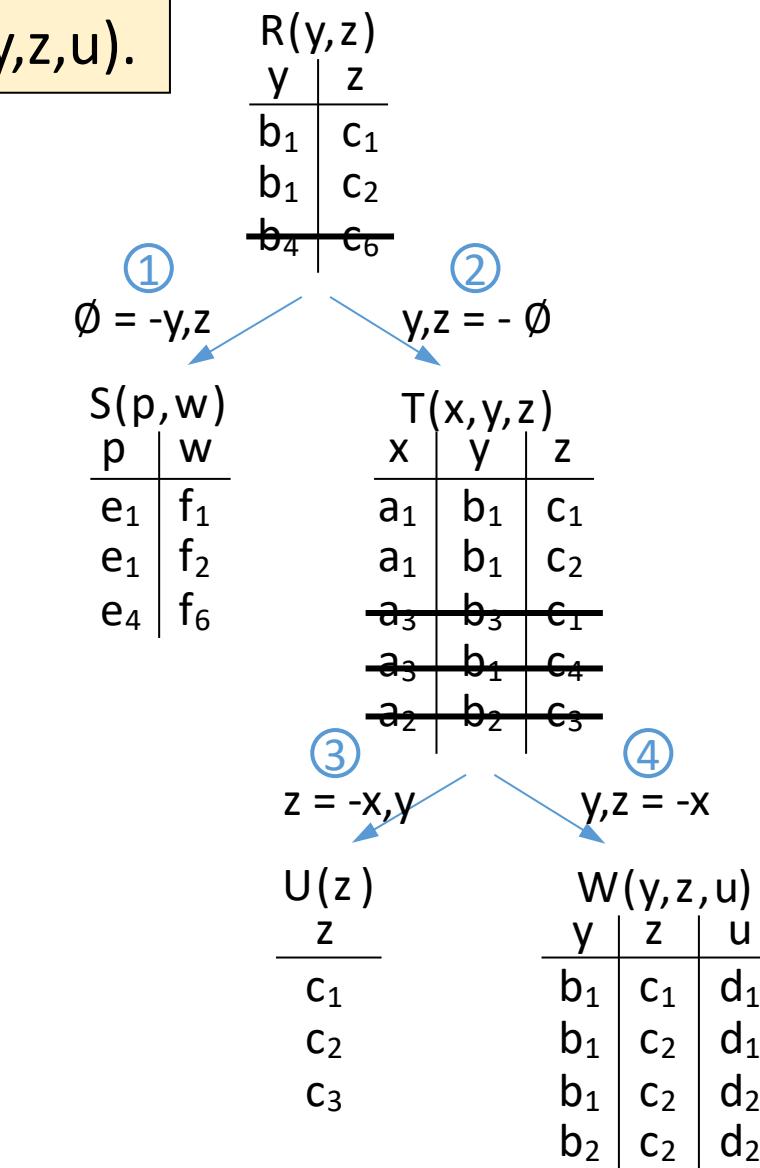
Yannakakis Algorithm example: 2nd pass

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

Semi-join phase \bowtie (remove dangling tuples) in $O(IN)$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order

2. Top-down semi-join propagation from root to leaves in some topological order



Yannakakis Algorithm example: 2nd pass

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

Semi-join phase ✕ (remove dangling tuples) in $O(IN)$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order
2. Top-down semi-join propagation from root to leaves in some topological order

The diagram shows the semi-join phase of the Yannakakis algorithm. It consists of five relations and their propagation steps:

- R(y,z)**: A relation with columns y and z. Rows are b₁, b₁, b₁, b₄. The last row b₄ is crossed out with a red line.
- T(x,y,z)**: A relation with columns x, y, z. Rows are a₁, a₁, a₃, a₃, a₃, a₂. The rows a₃, a₃, a₃, and a₂ are crossed out with a red line.
- S(p,w)**: A relation with columns p and w. Rows are e₁, e₁, e₄. The row e₄ is crossed out with a red line.
- U(z)**: A relation with column z. Rows are c₁, c₂, c₃.
- W(y,z,u)**: A relation with columns y, z, u. Rows are b₁, b₁, b₁, b₂. The rows b₁, b₁, b₁ are crossed out with a red line.

Annotations with arrows indicate the propagation of dangling tuples:

- From R(y,z) to T(x,y,z): A red arrow labeled "T~~x,y,z~~" points from the crossed-out row b₄ to the crossed-out row a₃. A blue circle labeled ① is near the arrow.
- From T(x,y,z) to R(y,z): A blue arrow labeled "y,z = -∅" points from the crossed-out row a₃ back to the crossed-out row b₄. A blue circle labeled ② is near the arrow.
- From S(p,w) to T(x,y,z): A red arrow labeled "∅ = -y,z" points from the crossed-out row e₄ to the crossed-out row a₃. A red circle with a slash is near the arrow.
- From T(x,y,z) to S(p,w): A blue arrow labeled "y,z = -∅" points from the crossed-out row a₃ back to the crossed-out row e₄. A blue circle labeled ③ is near the arrow.
- From T(x,y,z) to U(z): A blue arrow labeled "z = -x,y" points from the crossed-out row a₃ to the row c₁. A blue circle labeled ④ is near the arrow.
- From T(x,y,z) to W(y,z,u): A blue arrow labeled "y,z = -x" points from the crossed-out row a₃ to the crossed-out rows b₁, b₁, b₁. A blue circle labeled ④ is near the arrow.

Yannakakis Algorithm example: 2nd pass

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

Semi-join phase ✕ (remove dangling tuples) in $O(IN)$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order
2. Top-down semi-join propagation from root to leaves in some topological order

The diagram illustrates the execution of the Yannakakis algorithm across four stages:

- Stage 1 (Top):** A semi-join between $R(y,z)$ and $\emptyset = -y,z$. The result is $y,z = -\emptyset$, indicated by a crossed-out tuple (y, z) .
- Stage 2 (Middle Left):** A semi-join between $S(p,w)$ and $y,z = -\emptyset$. The result is $p,w = -y,z$.
- Stage 3 (Middle Right):** A semi-join between $T(x,y,z)$ and $p,w = -y,z$. The result is $x,y,z = -x$, indicated by crossed-out tuples (x, y, z) and (y, z) .
- Stage 4 (Bottom):** A semi-join between $U(z)$ and $x,y,z = -x$. The result is $z = -x,y$.

The tables show the state of each relation after each stage:

- $R(y,z)$:

| | |
|-------|-----------------------------|
| y | z |
| b_1 | c_1 |
| b_1 | c_2 |
| b_4 | c_6 |
- $S(p,w)$:

| | |
|-------|-------|
| p | w |
| e_1 | f_1 |
| e_1 | f_2 |
| e_4 | f_6 |
- $T(x,y,z)$:

| | | |
|-----------------------------|-----------------------------|-----------------------------|
| x | y | z |
| a_1 | b_1 | c_1 |
| a_1 | b_1 | c_2 |
| a_3 | b_3 | c_1 |
| a_3 | b_1 | c_4 |
| a_2 | b_2 | c_3 |
- $U(z)$:

| |
|-------|
| z |
| c_1 |
| c_2 |
| c_3 |
- $W(y,z,u)$:

| | | |
|-------|-------|-------|
| y | z | u |
| b_1 | c_1 | d_1 |
| b_1 | c_2 | d_1 |
| b_1 | c_2 | d_2 |
| b_2 | c_2 | d_2 |

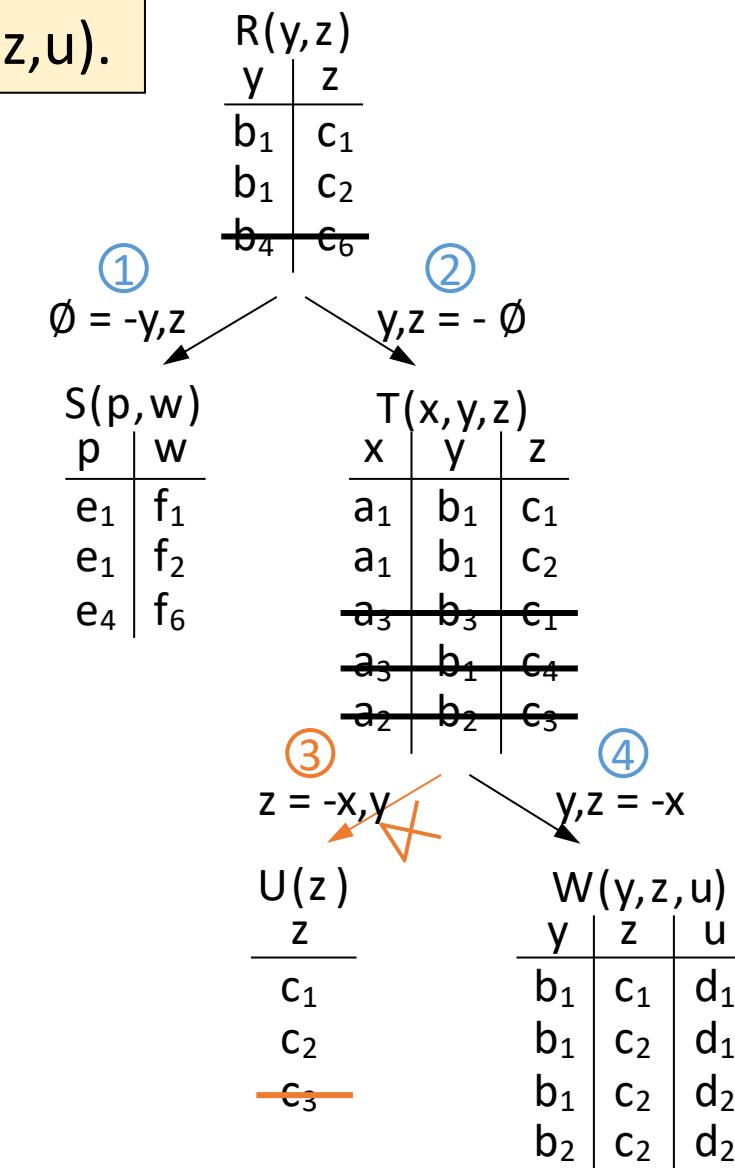
Yannakakis Algorithm example: 2nd pass

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

Semi-join phase ✕ (remove dangling tuples) in $O(IN)$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order

2. Top-down semi-join propagation from root to leaves in some topological order



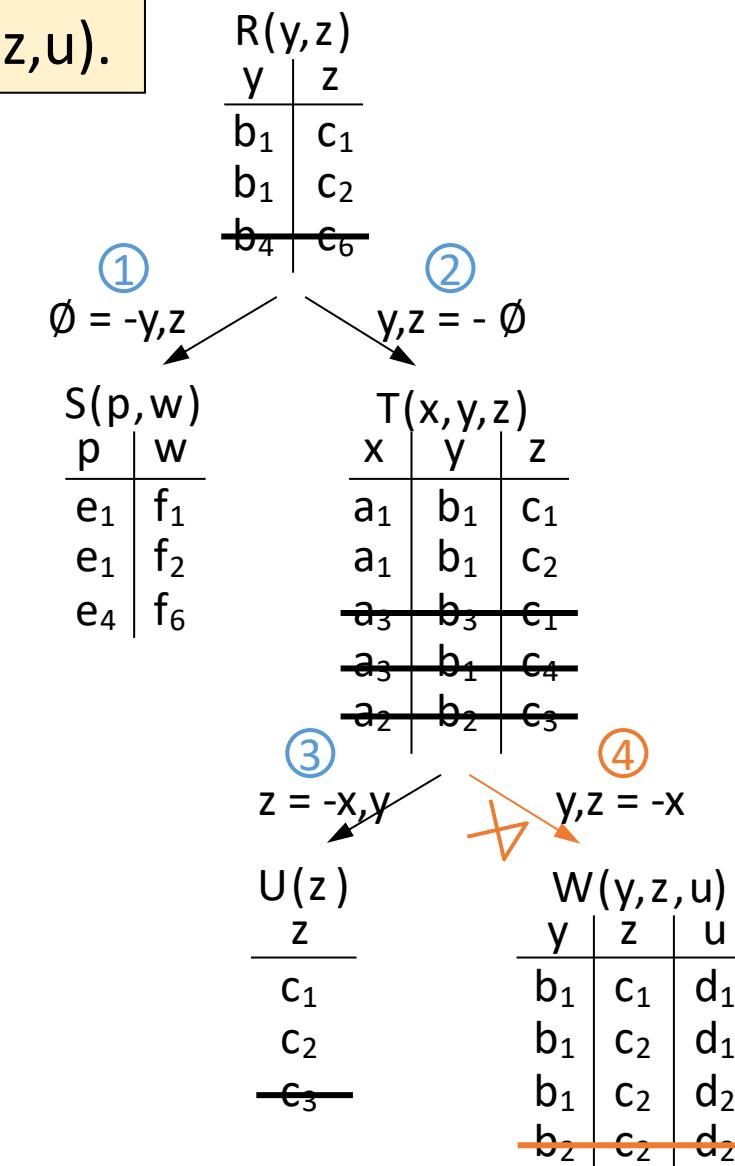
Yannakakis Algorithm example: 2nd pass

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

Semi-join phase ✕ (remove dangling tuples) in $O(IN)$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order

2. Top-down semi-join propagation from root to leaves in some topological order



Yannakakis Algorithm example: 2nd pass

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

Semi-join phase ✕ (remove dangling tuples) in $O(IN)$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order

2. Top-down semi-join propagation from root to leaves in some topological order

Notice that at the end of the second pass, all tables are reduced; no table contains any more dangling tuples.

In other words, *every* table now "knows" whether the Boolean version of the query is true.

The diagram illustrates the semi-join phase of the Yannakakis algorithm across four tables:

- R(y,z)**:
y | z
---|---
b₁ | c₁
b₁ | c₂
~~b₄~~ | ~~c₆~~
- S(p,w)**:
p | w
---|---
e₁ | f₁
e₁ | f₂
e₄ | f₆
- T(x,y,z)**:
x | y | z
---|---|---
a₁ | b₁ | c₁
a₁ | b₁ | c₂
~~a₃~~ | ~~b₃~~ | ~~c₁~~
~~a₃~~ | ~~b₁~~ | ~~c₄~~
~~a₂~~ | ~~b₂~~ | ~~c₃~~
- U(z)**:
z

c₁
c₂
~~c₃~~
- W(y,z,u)**:
y | z | u
---|---|---
b₁ | c₁ | d₁
b₁ | c₂ | d₁
b₁ | c₂ | d₂
~~b₂~~ | ~~c₂~~ | ~~d₂~~

Arrows indicate the flow of semi-join propagation:

- From $\emptyset = -y, z$ to $R(y,z)$ and $S(p,w)$.
- From $y, z = -\emptyset$ to $T(x,y,z)$.
- From $z = -x, y$ to $U(z)$ and $W(y,z,u)$.
- From $y, z = -x$ to $T(x,y,z)$.

Yannakakis Algorithm example: 3rd pass

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

Semi-join phase \bowtie (remove dangling tuples) in $O(\text{IN})$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order

2. Top-down semi-join propagation from root to leaves in some topological order

Join phase \bowtie (compute results) in $O(\text{OUT})$

3. Compute the results in a 2nd top-down (or 2nd bottom-up) traversal:

- This step can actually be combined with the earlier top-down traversal; thus two total passes (first from leaves, then from root) are actually enough ☺

Notice how with every join, the join result can never decrease in size!

| R(y,z) | |
|--------------------------|--------------------------|
| y | z |
| b ₁ | c ₁ |
| b ₁ | c ₂ |
| b₄ | c₆ |

| S(p,w) | |
|----------------|----------------|
| p | w |
| e ₁ | f ₁ |
| e ₁ | f ₂ |
| e ₄ | f ₆ |

| T(x,y,z) | | |
|--------------------------|--------------------------|--------------------------|
| x | y | z |
| a ₁ | b ₁ | c ₁ |
| a ₁ | b ₁ | c ₂ |
| a₃ | b₃ | c₁ |
| a₃ | b₁ | c₄ |
| a₂ | b₂ | c₃ |

| U(z) |
|--------------------------|
| z |
| c ₁ |
| c ₂ |
| c₃ |

| W(y,z,u) | | |
|--------------------------|--------------------------|--------------------------|
| y | z | u |
| b ₁ | c ₁ | d ₁ |
| b ₁ | c ₂ | d ₁ |
| b ₁ | c ₂ | d ₂ |
| b₂ | c₂ | d₂ |

Join results

?

Yannakakis Algorithm example: 3rd pass

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

Semi-join phase \bowtie (remove dangling tuples) in $O(\text{IN})$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order

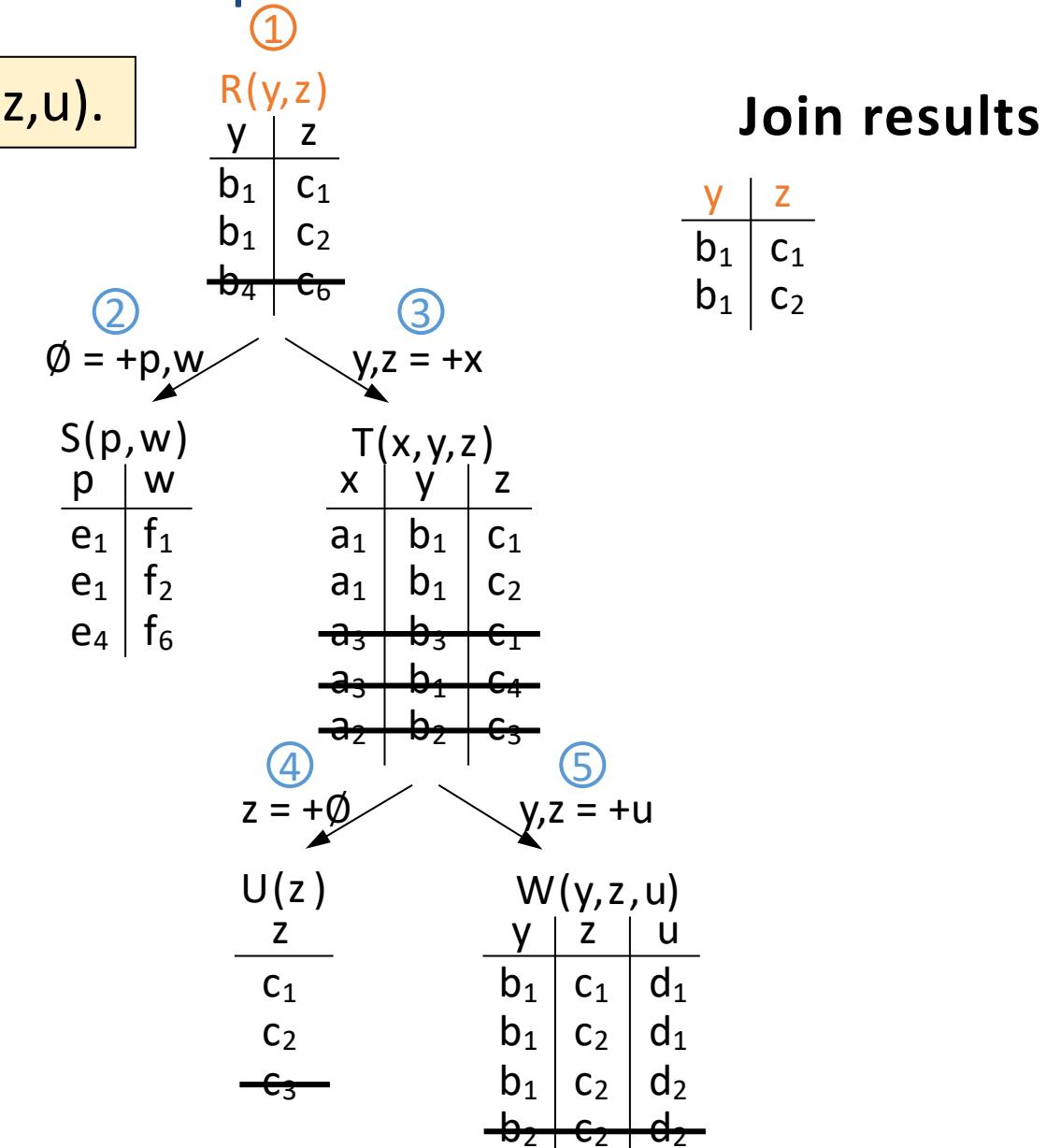
2. Top-down semi-join propagation from root to leaves in some topological order

Join phase \bowtie (compute results) in $O(\text{OUT})$

3. Compute the results in a 2nd top-down (or 2nd bottom-up) traversal:

- This step can actually be combined with the earlier top-down traversal; thus two total passes (first from leaves, then from root) are actually enough ☺

Notice how with every join, the join result can never decrease in size!



Yannakakis Algorithm example: 3rd pass

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

Semi-join phase \bowtie (remove dangling tuples) in $O(\text{IN})$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order
2. Top-down semi-join propagation from root to leaves in some topological order

Join phase \bowtie (compute results) in $O(\text{OUT})$

3. Compute the results in a 2nd top-down (or 2nd bottom-up) traversal:

- This step can actually be combined with the earlier top-down traversal; thus two total passes (first from leaves, then from root) are actually enough ☺

Notice how with every join, the join result can never decrease in size!

Join results

| y | z | p | w |
|----------------|----------------|----------------|----------------|
| b ₁ | c ₁ | e ₁ | f ₁ |
| b ₁ | c ₁ | e ₁ | f ₂ |
| b ₁ | c ₁ | e ₄ | f ₆ |
| b ₁ | c ₂ | e ₁ | f ₁ |
| b ₁ | c ₂ | e ₁ | f ₂ |
| b ₁ | c ₂ | e ₄ | f ₆ |

Yannakakis Algorithm example: 3rd pass

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

Semi-join phase \bowtie (remove dangling tuples) in $O(\text{IN})$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order
2. Top-down semi-join propagation from root to leaves in some topological order

Join phase \bowtie (compute results) in $O(\text{OUT})$

3. Compute the results in a 2nd top-down (or 2nd bottom-up) traversal:

- This step can actually be combined with the earlier top-down traversal; thus two total passes (first from leaves, then from root) are actually enough ☺

Notice how with every join, the join result can never decrease in size!

The diagram illustrates the Yannakakis algorithm's 3rd pass through five stages:

- Stage 1 (Semi-join phase):** Shows the initial relations $R(y,z)$, $S(p,w)$, $T(x,y,z)$, $U(z)$, and $W(y,z,u)$. Dangling tuples are crossed out: b_4 in R , e_6 in S , a_3 in T , c_3 in U , and d_2 in W .
- Stage 2 (Top-down semi-join propagation):** The set $\emptyset = +p,w$ is propagated up. A join condition $y,z = +x$ is shown between $S(p,w)$ and $T(x,y,z)$.
- Stage 3 (Bottom-up semi-join propagation):** The set $T(x,y,z)$ is propagated down. A join condition $y,z = +u$ is shown between $T(x,y,z)$ and $W(y,z,u)$.
- Stage 4 (Join phase):** The set $z = +\emptyset$ is propagated down. A join condition $y,z = +u$ is shown between $U(z)$ and $W(y,z,u)$.
- Stage 5 (Final results):** The final join results are shown in a table with columns y, z, p, w, x, u . The rows are:

| y | z | p | w | x |
|-------|-------|-------|-------|-------|
| b_1 | c_1 | e_1 | f_1 | a_1 |
| b_1 | c_1 | e_1 | f_2 | a_1 |
| b_1 | c_1 | e_4 | f_6 | a_1 |
| b_1 | c_2 | e_1 | f_1 | a_1 |
| b_1 | c_2 | e_1 | f_2 | a_1 |
| b_1 | c_2 | e_4 | f_6 | a_1 |

Yannakakis Algorithm example: 3rd pass

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

Semi-join phase \bowtie (remove dangling tuples) in $O(\text{IN})$

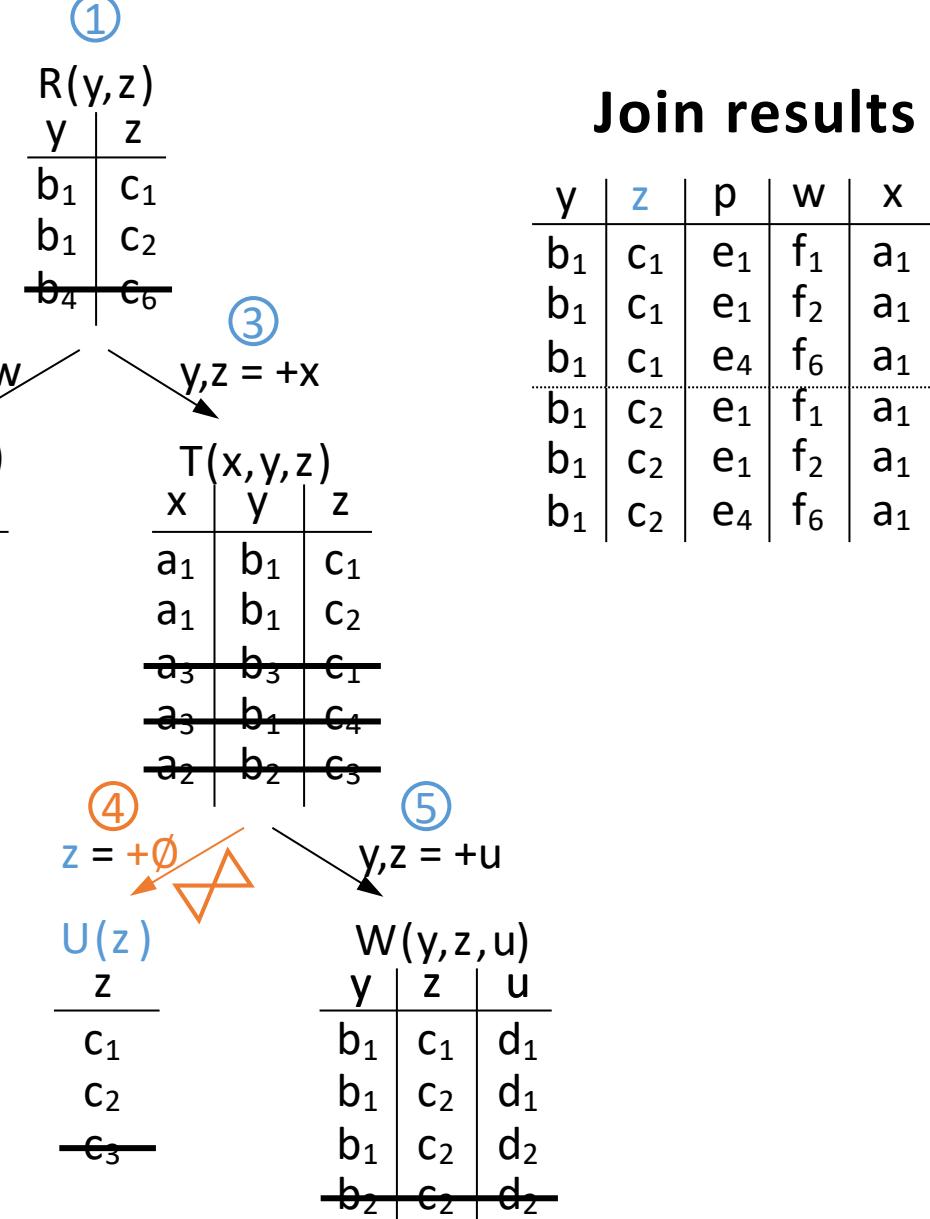
1. Bottom-up semi-join propagation from leaves to root in some reverse topological order
2. Top-down semi-join propagation from root to leaves in some topological order

Join phase \bowtie (compute results) in $O(\text{OUT})$

3. Compute the results in a 2nd top-down (or 2nd bottom-up) traversal:

- This step can actually be combined with the earlier top-down traversal; thus two total passes (first from leaves, then from root) are actually enough ☺

Notice how with every join, the join result can never decrease in size!



Yannakakis Algorithm example: 3rd pass

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

Semi-join phase \bowtie (remove dangling tuples) in $O(\text{IN})$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order

2. Top-down semi-join propagation from root to leaves in some topological order

Join phase \bowtie (compute results) in $O(\text{OUT})$

3. Compute the results in a 2nd top-down (or 2nd bottom-up) traversal:

- This step can actually be combined with the earlier top-down traversal; thus two total passes (first from leaves, then from root) are actually enough ☺

Notice how with every join, the join result can never decrease in size!

①

$R(y,z)$

| | |
|-------|-----------------------------|
| y | z |
| b_1 | c_1 |
| b_1 | c_2 |
| b_4 | e_6 |

②

$S(p,w)$

| | |
|-------|-------|
| p | w |
| e_1 | f_1 |
| e_1 | f_2 |
| e_4 | f_6 |

③

$T(x,y,z)$

| | | |
|-----------------------------|-----------------------------|-----------------------------|
| x | y | z |
| a_1 | b_1 | c_1 |
| a_1 | b_1 | c_2 |
| a_3 | b_3 | c_1 |
| a_3 | b_1 | c_4 |
| a_2 | b_2 | c_3 |

④

$U(z)$

| |
|-----------------------------|
| z |
| c_1 |
| c_2 |
| c_3 |

⑤

$W(y,z,u)$

| | | |
|-----------------------------|-----------------------------|-----------------------------|
| y | z | u |
| b_1 | c_1 | d_1 |
| b_1 | c_2 | d_1 |
| b_1 | c_2 | d_2 |
| b_2 | c_2 | d_2 |

Join results

| y | z | p | w | x | u |
|-------|-------|-------|-------|-------|-------|
| b_1 | c_1 | e_1 | f_1 | a_1 | d_1 |
| b_1 | c_1 | e_1 | f_2 | a_1 | d_1 |
| b_1 | c_1 | e_4 | f_6 | a_1 | d_1 |
| b_1 | c_2 | e_1 | f_1 | a_1 | d_1 |
| b_1 | c_2 | e_1 | f_2 | a_1 | d_1 |
| b_1 | c_2 | e_4 | f_6 | a_1 | d_1 |
| b_1 | c_2 | e_1 | f_1 | a_1 | d_2 |
| b_1 | c_2 | e_1 | f_2 | a_1 | d_2 |
| b_1 | c_2 | e_4 | f_6 | a_1 | d_2 |

Yannakakis Algorithm example: summary

$Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).$

Semi-join phase ✖ (remove dangling tuples) in $O(\text{IN})$

1. Bottom-up semi-join propagation from leaves to root in some reverse topological order
2. Top-down semi-join propagation from root to leaves in some topological order

Join phase ⚡ (compute results) in $O(\text{OUT})$

3. Compute the results in a 2nd top-down (or 2nd bottom-up) traversal:

- This step can actually be combined with the earlier top-down traversal; thus two total passes (first from leaves, then from root) are actually enough ☺

| $R(y,z)$ | |
|----------------|----------------|
| y | z |
| b_1 | c_1 |
| b_1 | c_2 |
| $\cancel{b_4}$ | $\cancel{e_6}$ |

| \emptyset | | y,z | | |
|-------------|-------|----------------|----------------|----------------|
| | | x | y | z |
| $S(p,w)$ | | | | |
| p | w | | | |
| e_1 | f_1 | a_1 | b_1 | c_1 |
| e_1 | f_2 | a_1 | b_1 | c_2 |
| e_4 | f_6 | $\cancel{a_3}$ | $\cancel{b_3}$ | $\cancel{c_1}$ |
| | | $\cancel{a_3}$ | $\cancel{b_1}$ | $\cancel{c_4}$ |
| | | $\cancel{a_2}$ | $\cancel{b_2}$ | $\cancel{c_3}$ |

| $U(z)$ | | $W(y,z,u)$ | | |
|----------------|--|----------------|----------------|----------------|
| z | | y | z | u |
| c_1 | | b_1 | c_1 | d_1 |
| c_2 | | b_1 | c_2 | d_1 |
| $\cancel{c_3}$ | | b_1 | c_2 | d_2 |
| | | $\cancel{b_2}$ | $\cancel{c_2}$ | $\cancel{d_2}$ |

Join results

| y | z | p | w | x | u |
|-------|-------|-------|-------|-------|-------|
| b_1 | c_1 | e_1 | f_1 | a_1 | d_1 |
| b_1 | c_1 | e_1 | f_2 | a_1 | d_1 |
| b_1 | c_1 | e_4 | f_6 | a_1 | d_1 |
| b_1 | c_2 | e_1 | f_1 | a_1 | d_1 |
| b_1 | c_2 | e_1 | f_2 | a_1 | d_1 |
| b_1 | c_2 | e_4 | f_6 | a_1 | d_1 |
| b_1 | c_2 | e_1 | f_1 | a_1 | d_2 |
| b_1 | c_2 | e_1 | f_2 | a_1 | d_2 |
| b_1 | c_2 | e_4 | f_6 | a_1 | d_2 |

Deciding ACQs Efficiently [Yannakakis'81]

- Non-Serial Dynamic Programming (NSDP) algorithm over a **join tree** $T = (V, E)$ of a query Q , given database instance D
- Decide Boolean variant $Q(D) = \emptyset$ as follows:
 - Pick a root and assign to each $R_j \in V$ the corresponding relation R_j^D of D
 - In a bottom-up reverse topological order of T : compute semijoins of R_j^D
 - If the thus reduced relation at root is empty, then $Q(D) = \emptyset$, else $Q(D) \neq \emptyset$.
- Theorem:
 - For ACQs Q : Deciding $Q(D) = \emptyset$ is feasible in INPUT linear time.
 - Computing $Q(D)$ can be done in OUTPUT polynomial time.
 - For full queries (no projections) in OUTPUT linear time.

Recap

- T2:
 - The notions and complexity of query equivalence and containment
 - The Homomorphism Theorem
 - Minimization of conjunctive queries
- T3 so far: Acyclic conjunctive queries
 - The Yannakakis algorithm
- T3 yet to be seen:
 - What do we do with cycles?
 - What about ranked retrieval?

Pointers to related work

- Abiteboul, Hull, Vianu. *Foundations of Databases*. Addison Wesley, 1995. Ch 6.4: Acyclic joins, semi-join reduction, Yannakakis, GYO algorithm. <http://webdam.inria.fr/Alice/>
- The original GYO algorithm was developed concurrently by Graham and Yu-Ozsoyoglu:
 - [Gra79] Graham. *On the universal relation*. Technical Report, University of Toronto, 1979.
 - [YO79] Yu, Ozsoyoglu. *An algorithm for tree-query membership of a distributed query*. COMPSAC, 1979.
- Yannakakis. *Algorithms for acyclic database schemes*. VLDB 1981
<https://dl.acm.org/doi/10.5555/1286831.1286840>
- Bernstein, Chiu. *Using semi-joins to solve relational queries*. JACM 1981.
<https://doi.org/10.1145/322234.322238>
- Bernstein, Goodman. *Power of natural semi-joins*. SIAM J. 1981.
<https://doi.org/10.1137/0210059>
- Beeri, Fagin, Maier, Yannakakis. *On the desirability of acyclic database schemes*. JACM 1983.
<https://doi.org/10.1145/2402.322389>