Updated 3/26/2022

Topic 3: Efficient query evaluation
Unit 1: Acyclic query evaluation
Lecture 18

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp22)

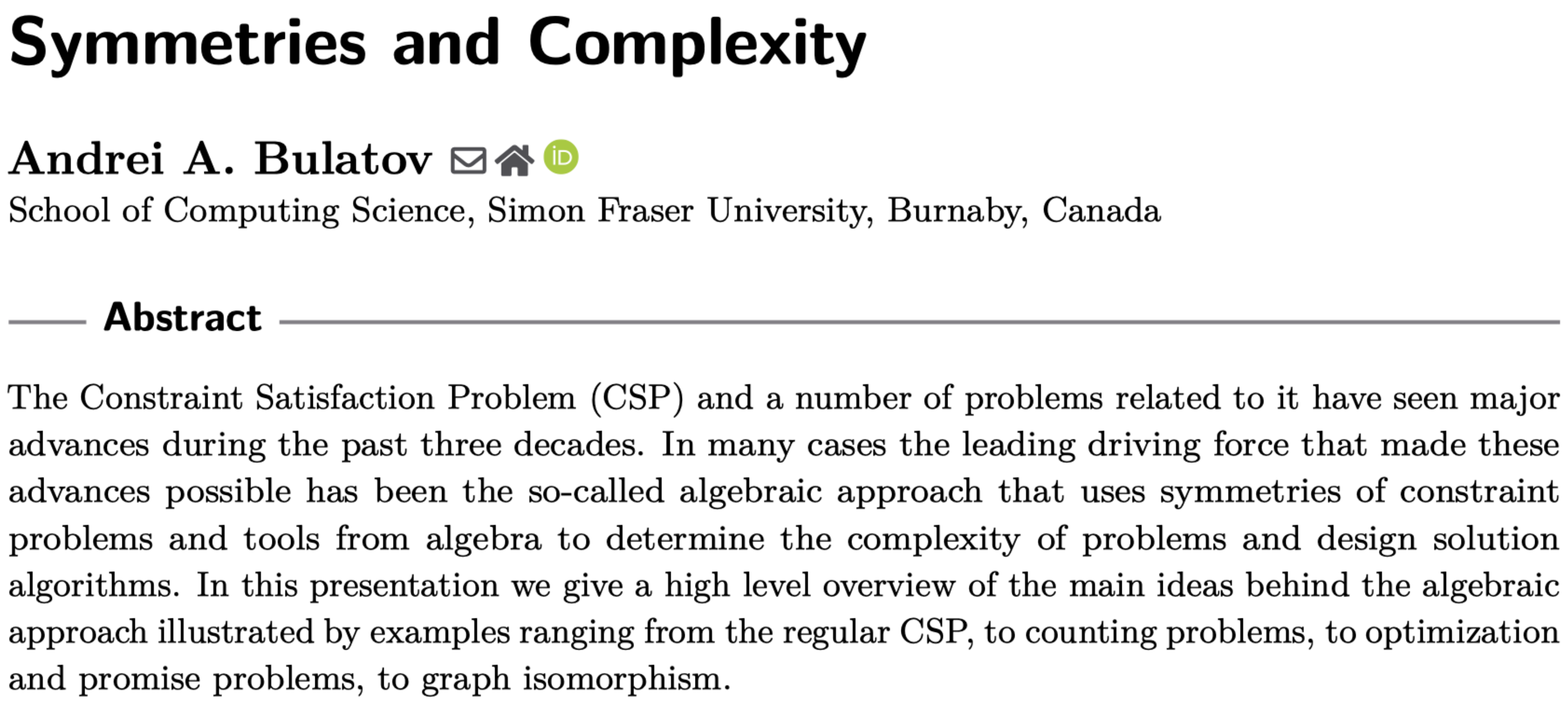
https://northeastern-datalab.github.io/cs7240/sp22/

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

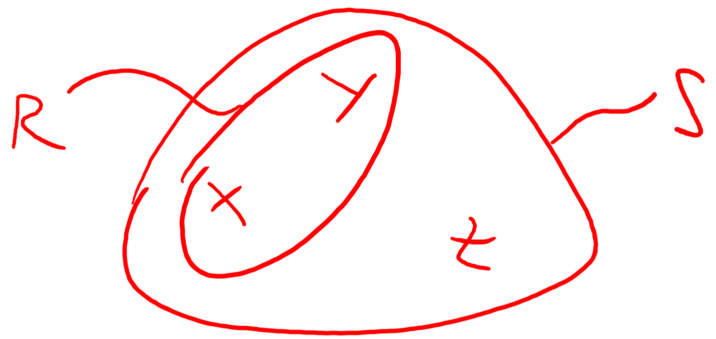
The Constraint Satisfaction Problem (CSP) and a number of problems related to it have seen major advances during the past three decades. In many cases the leading driving force that made these advances possible has been the so-called algebraic approach that uses symmetries of constraint problems and 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

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

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



Join tree

?

Query hypergraph

?

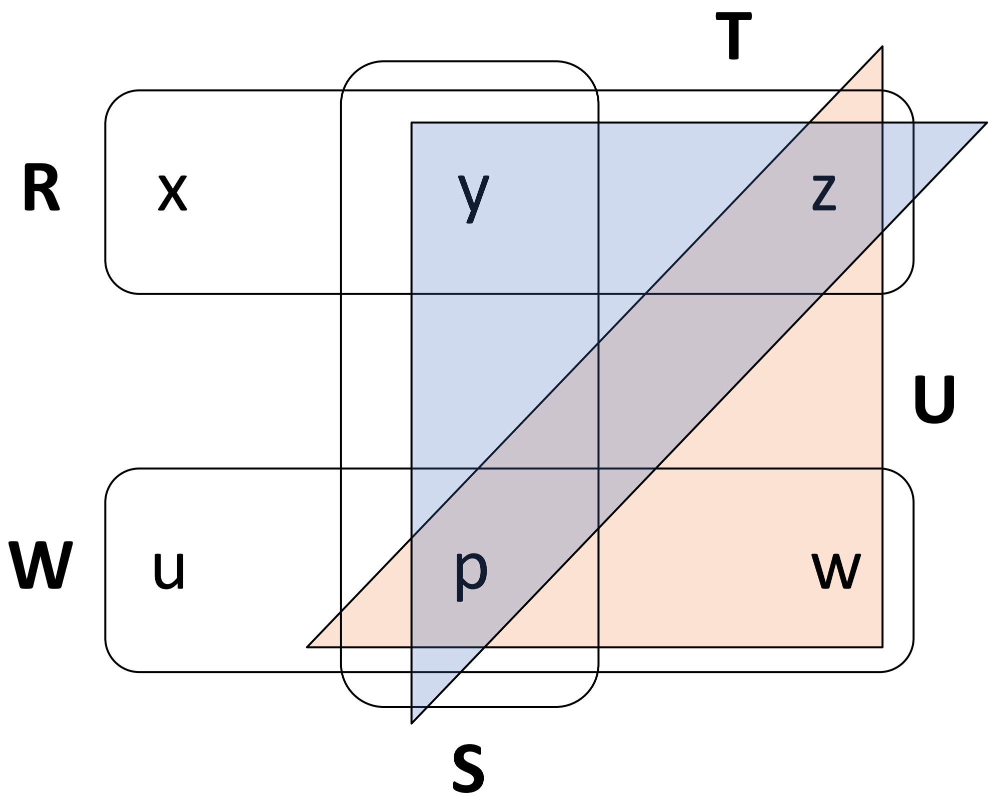
GYO reduction: Example 2

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

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

Join tree

?



GYO reduction: Example 2

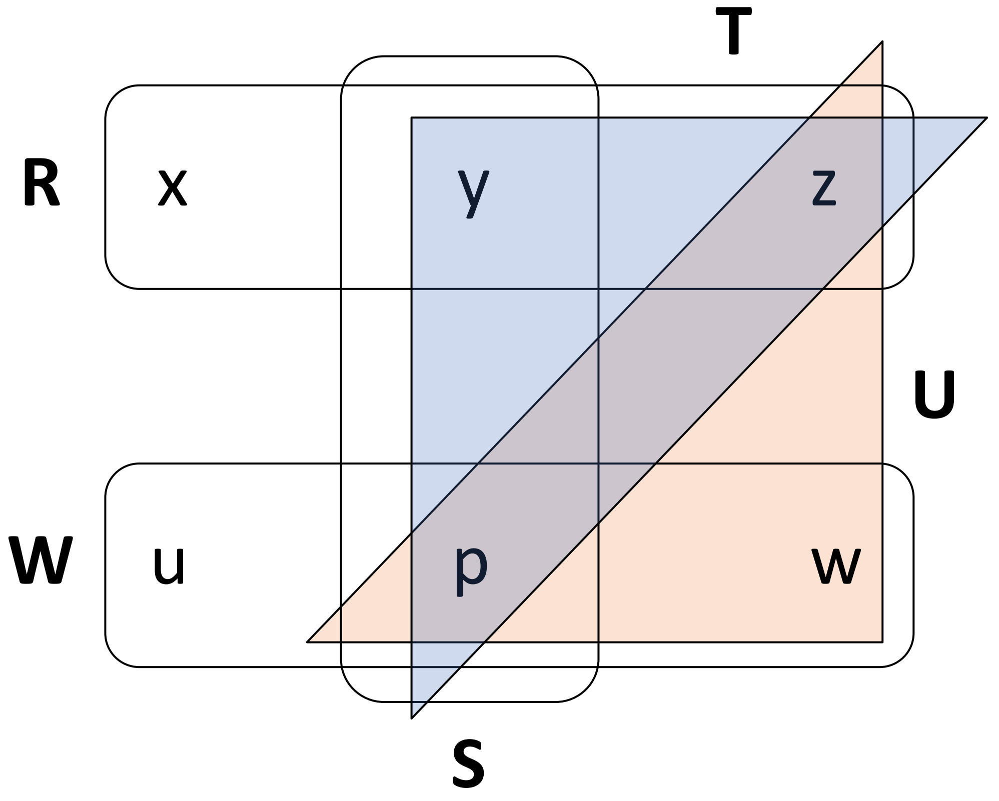
Q :- R(x,y,z), S(y,p), T(y,z,p), U(z,u,p), W(u,p,w).

- GYO Ear removal
  - 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

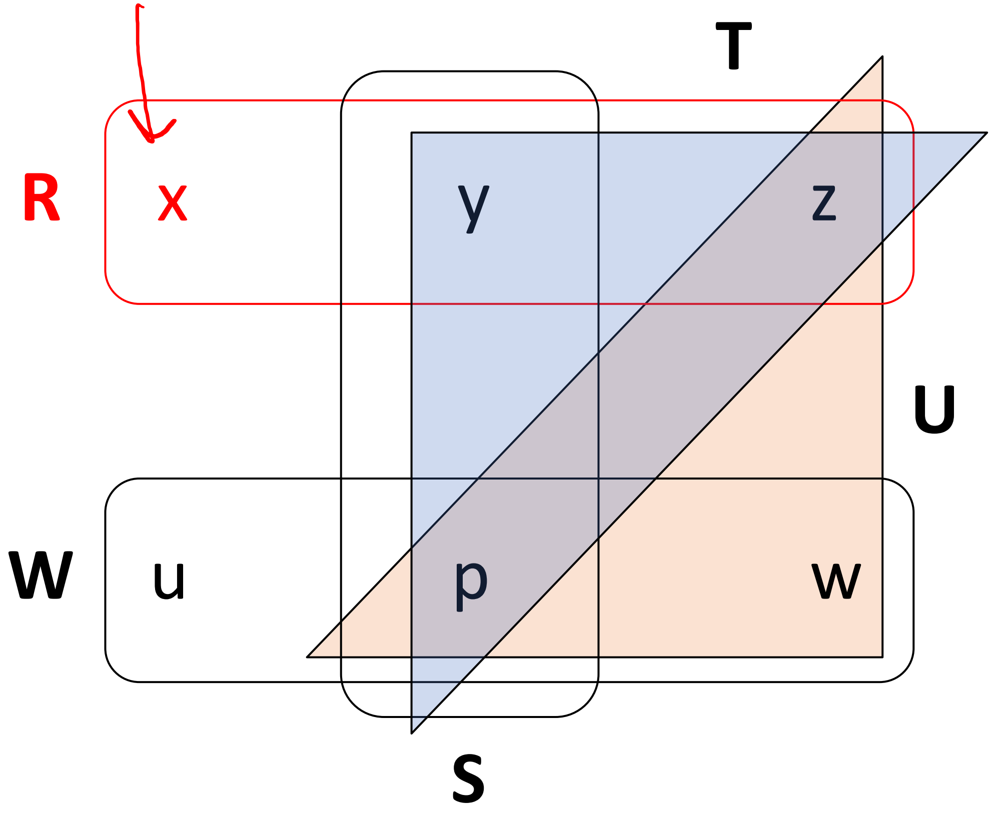
Q :- R(x,y,z), S(y,p), T(y,z,p), U(z,u,p), W(u,p,w).

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

U(z,p,w)

T(y,z,p)

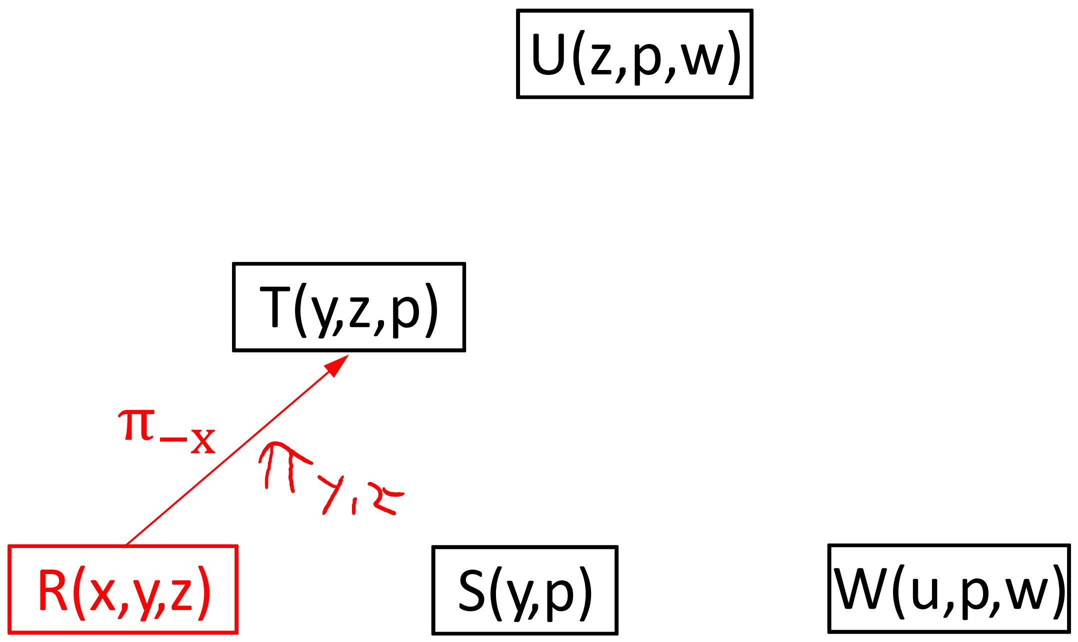
R(x,y,z) S(y,p) W(u,p,w)

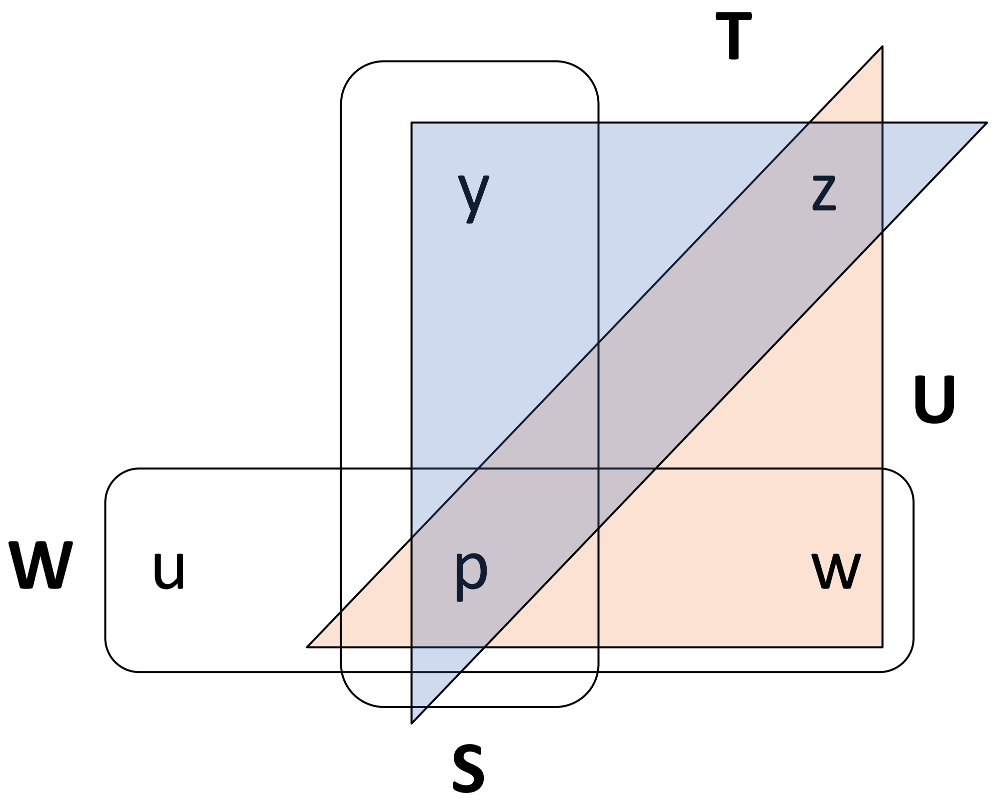


GYO reduction: Example 2

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

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

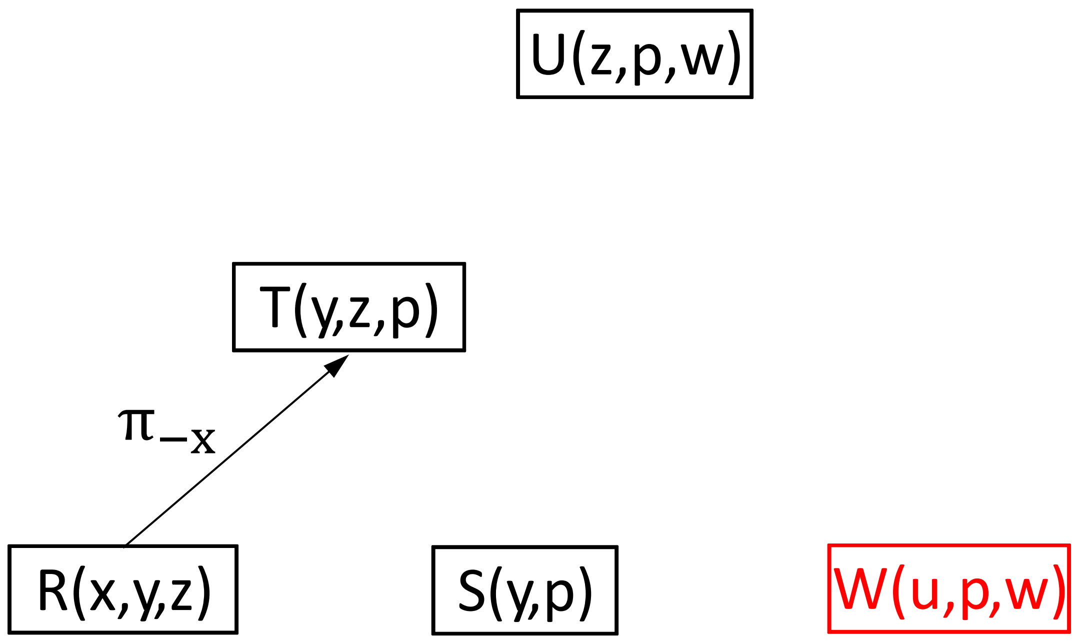


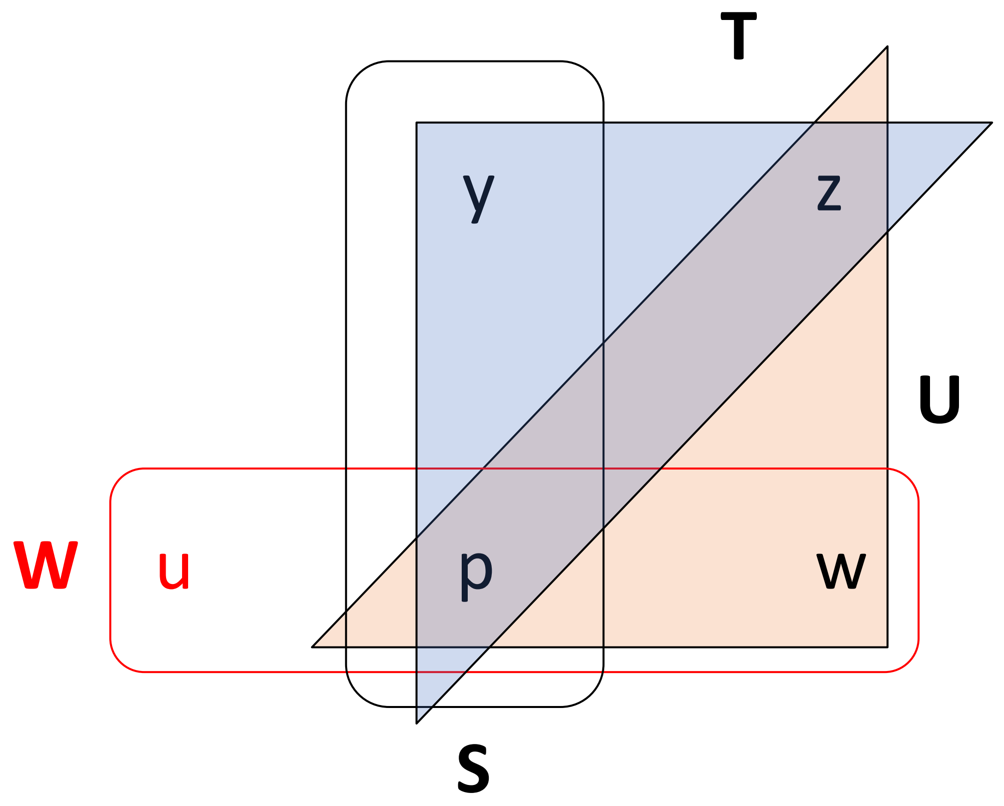


GYO reduction: Example 2

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

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

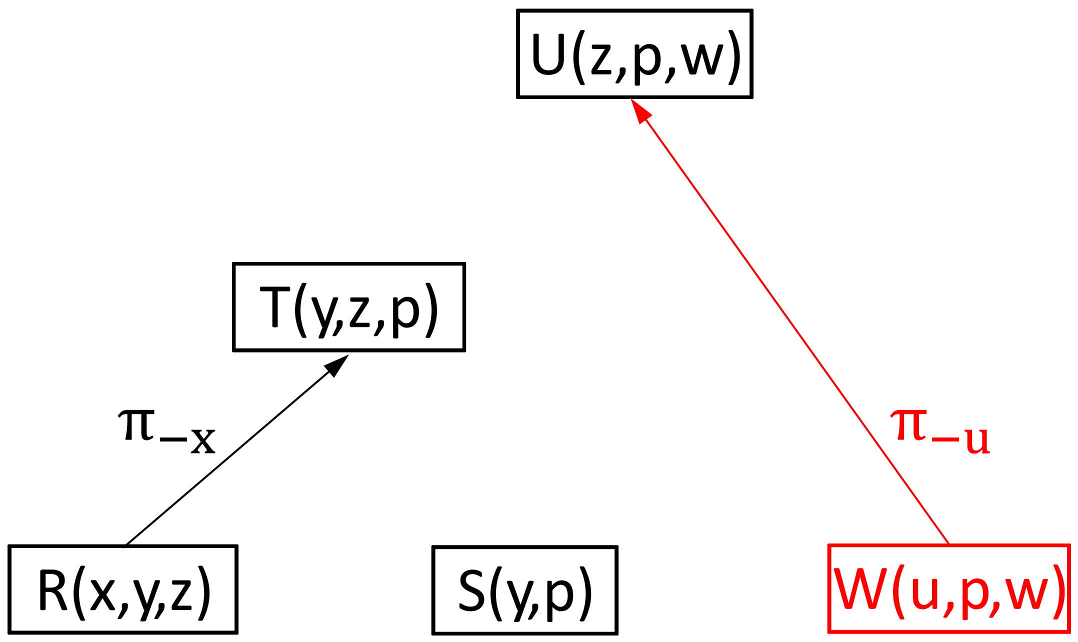


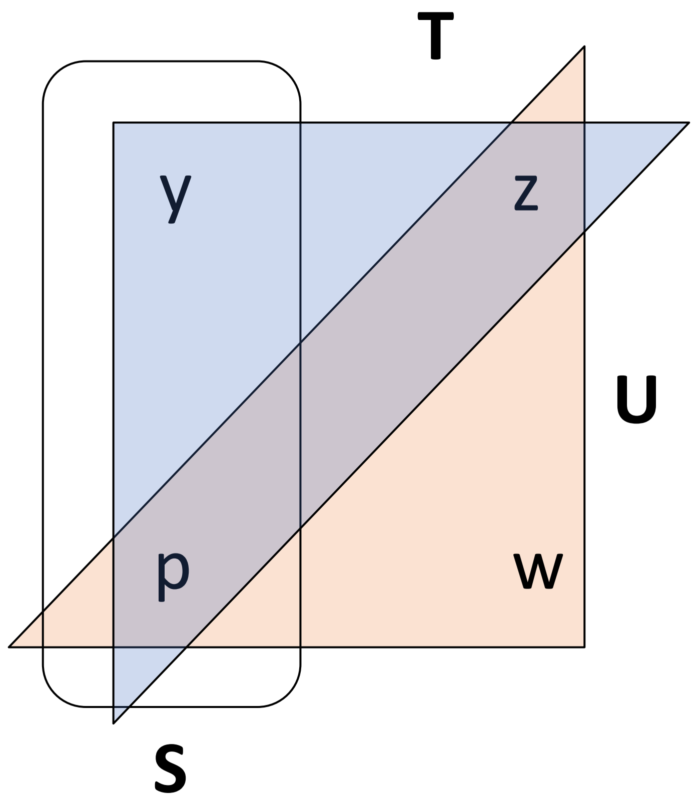


GYO reduction: Example 2

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

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



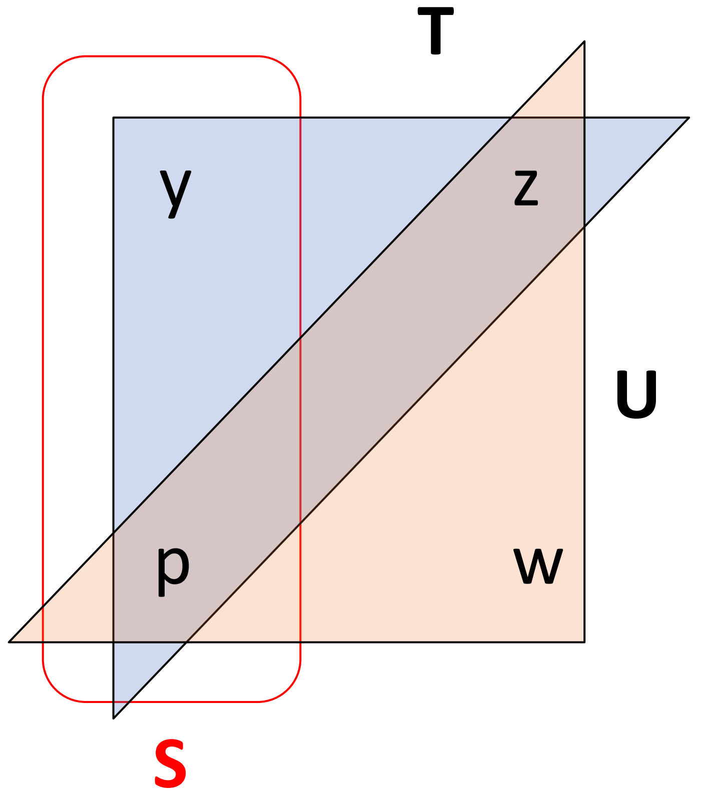


GYO reduction: Example 2

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

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

$$\begin{tikzcd}
 & & U(z,p,w) \\
 & T(y,z,p) & \\
R(x,y,z) \arrow[ur, "\pi_{-x}"] & S(y,p) & W(u,p,w) \arrow[uu, "\pi_{-u}"']
\end{tikzcd}$$

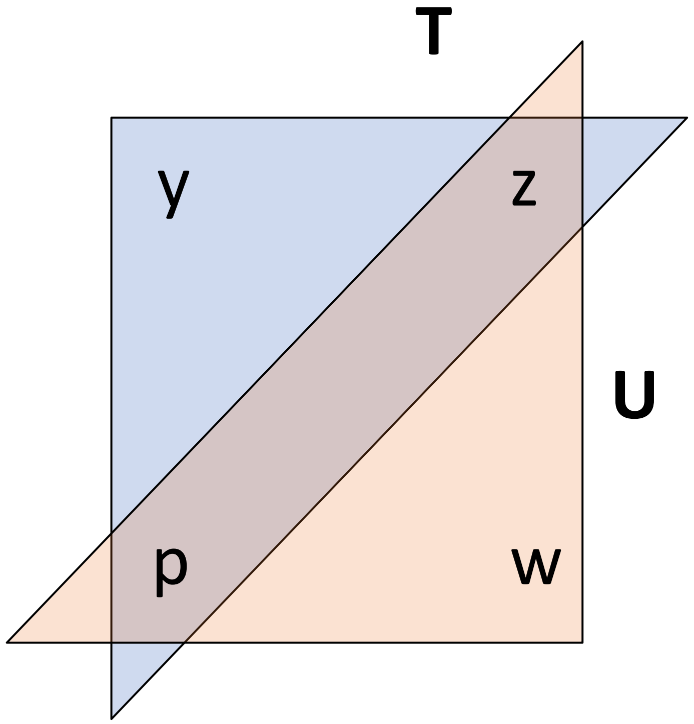


GYO reduction: Example 2

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

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

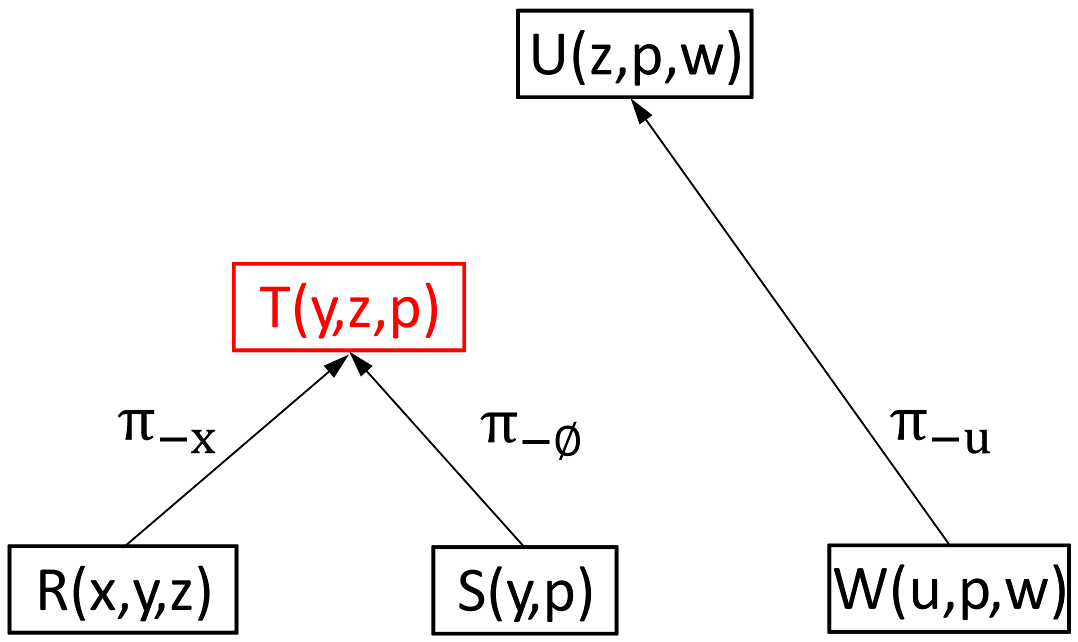
$$\begin{tikzcd}
 & & U(z,p,w) & \\
 & T(y,z,p) & & \\
R(x,y,z) \arrow[ur, "\pi_{-x}"] & S(y,p) \arrow[u, "\pi_{-\emptyset}"'] & & W(u,p,w) \arrow[uul, "\pi_{-u}"']
\end{tikzcd}$$

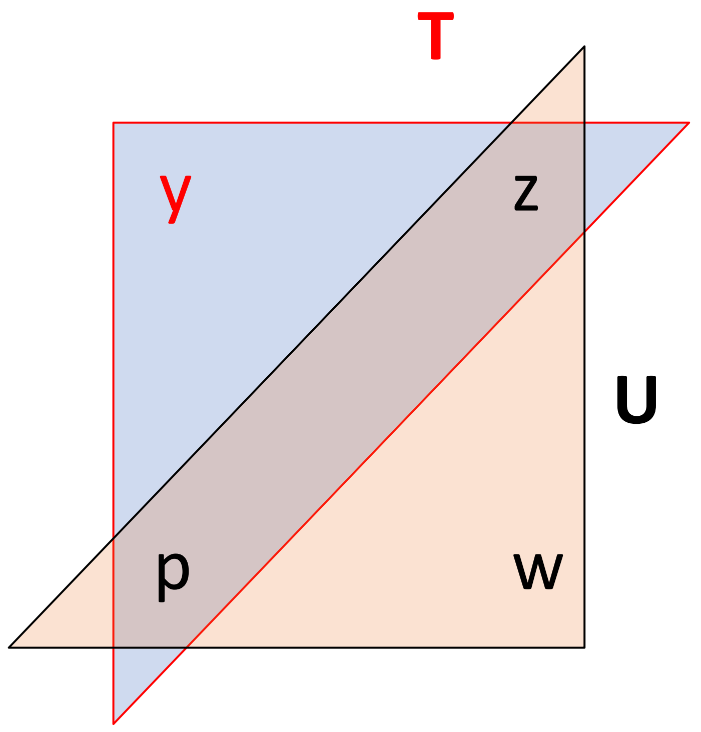


GYO reduction: Example 2

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

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

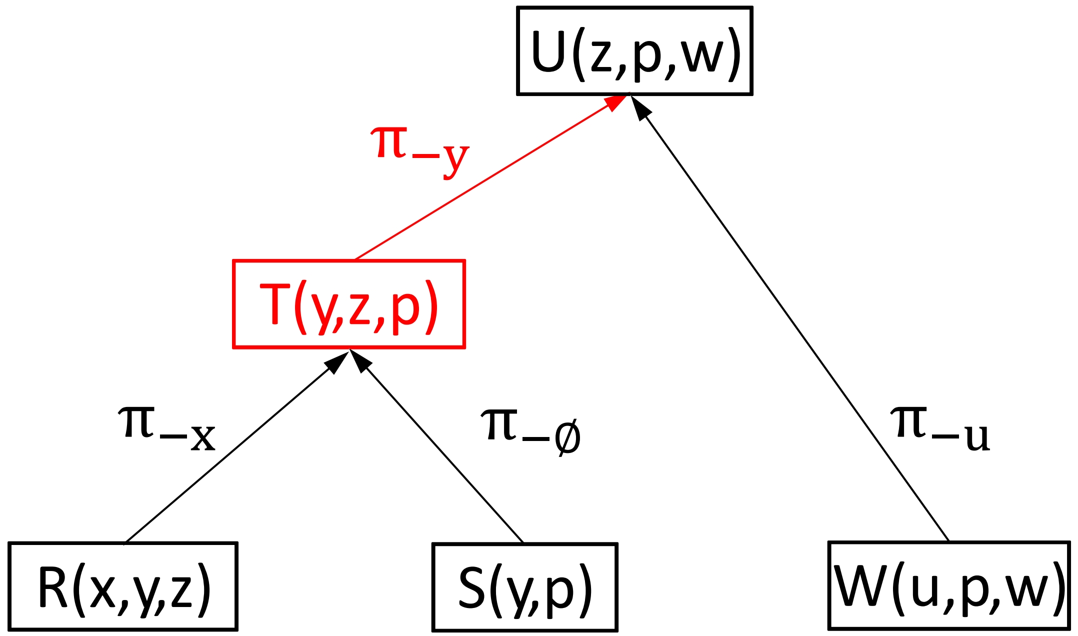


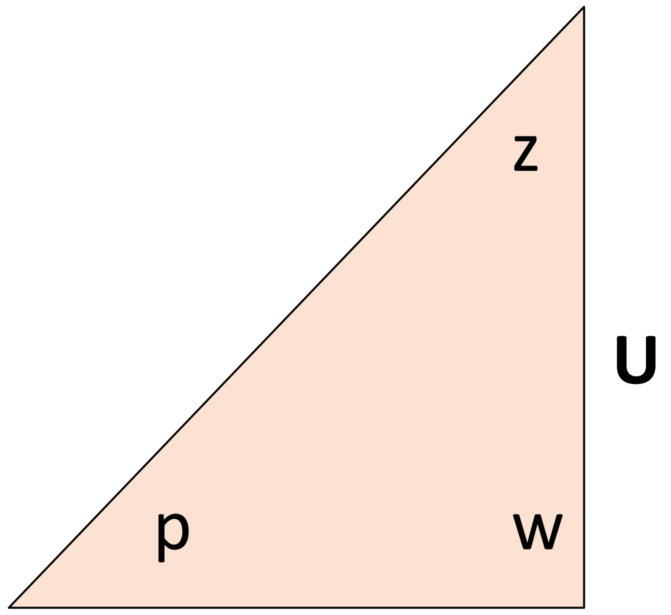


GYO reduction: Example 2

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

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

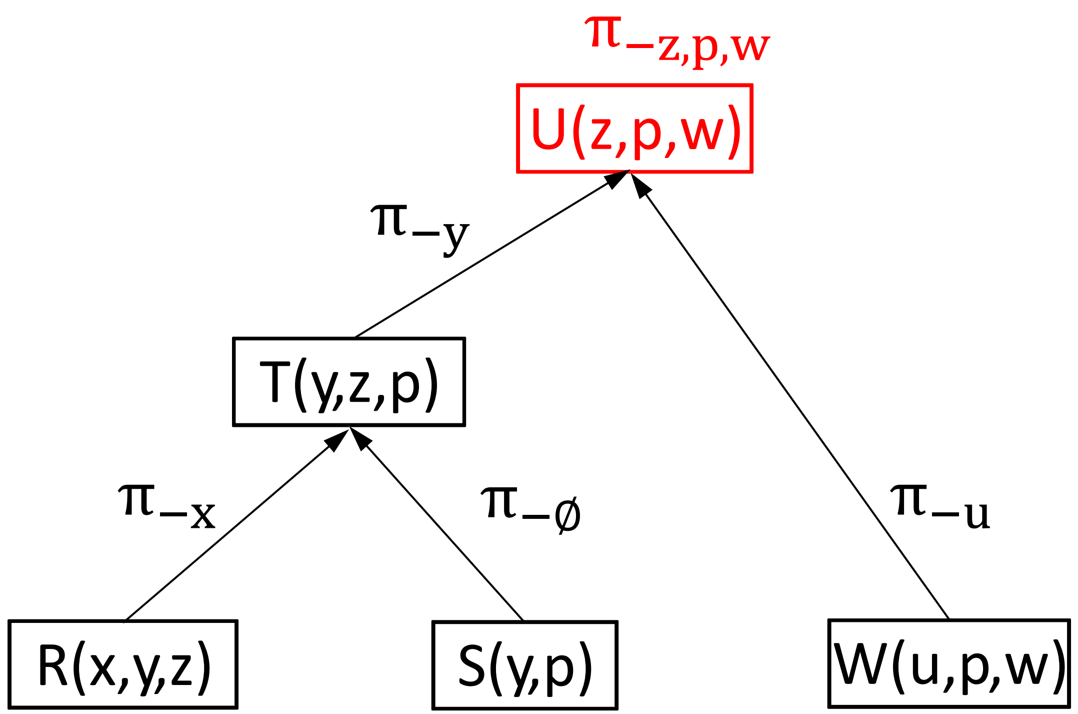


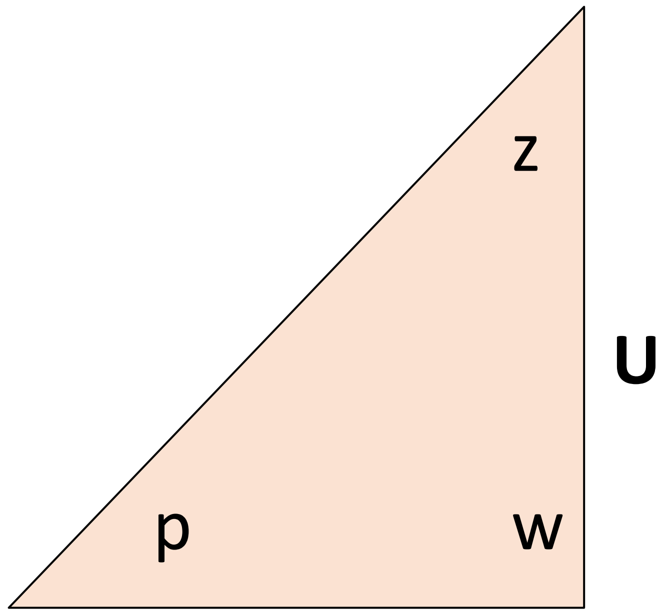


GYO reduction: Example 2

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

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



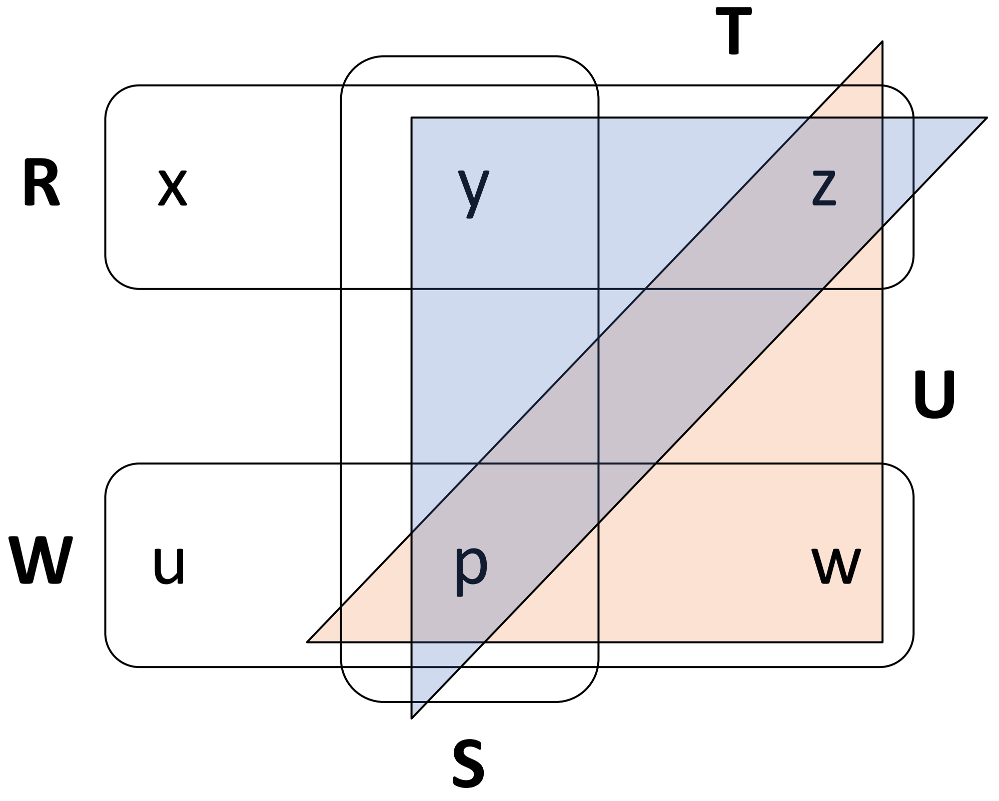


GYO reduction: Example 2

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

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

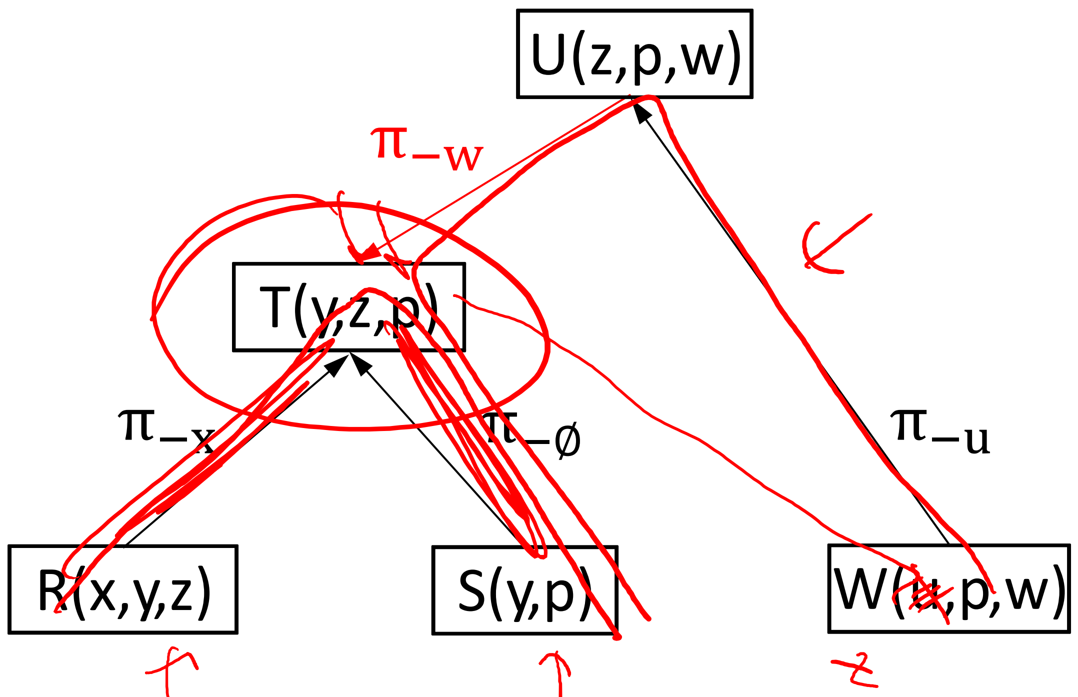
$$\begin{tikzcd}
 & & & \boxed{U(z,p,w)} & \\
 & \boxed{T(y,z,p)} \arrow[urr, "\pi_{-y}"] & & & \\
\boxed{R(x,y,z)} \arrow[ur, "\pi_{-x}"] & & \boxed{S(y,p)} \arrow[ul, "\pi_{-\emptyset}"'] & & \boxed{W(u,p,w)} \arrow[uul, "\pi_{-u}"']
\end{tikzcd}$$

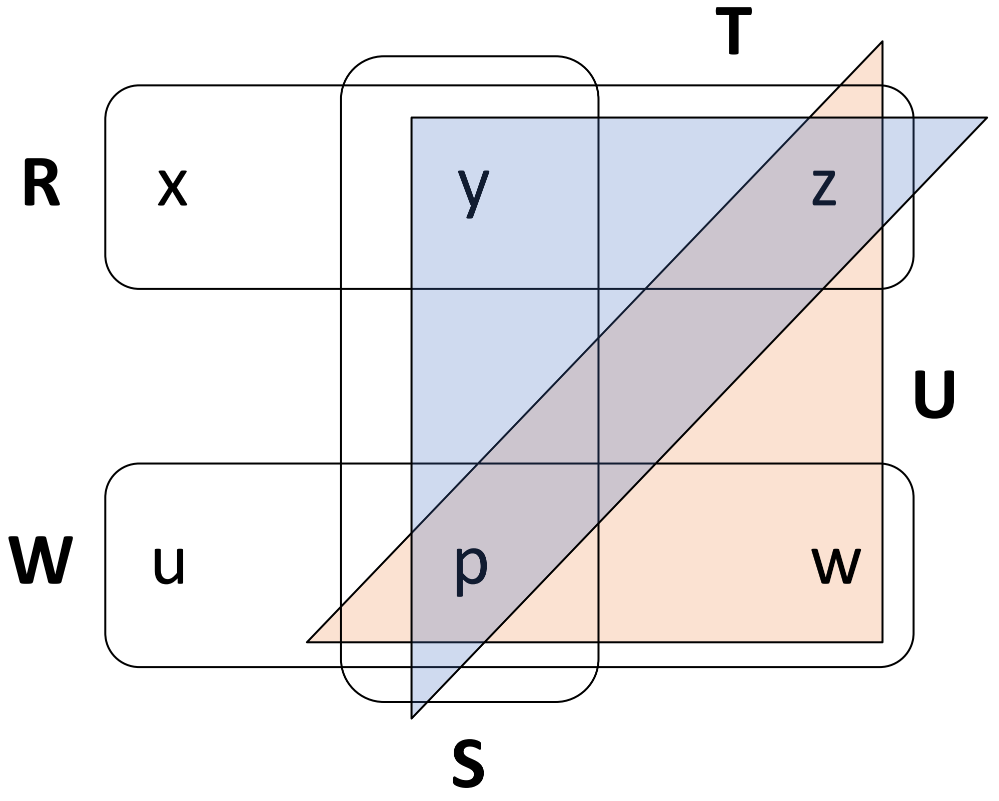


GYO reduction: Example 2

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

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





GYO reduction: Example 3: Triangle

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

- 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

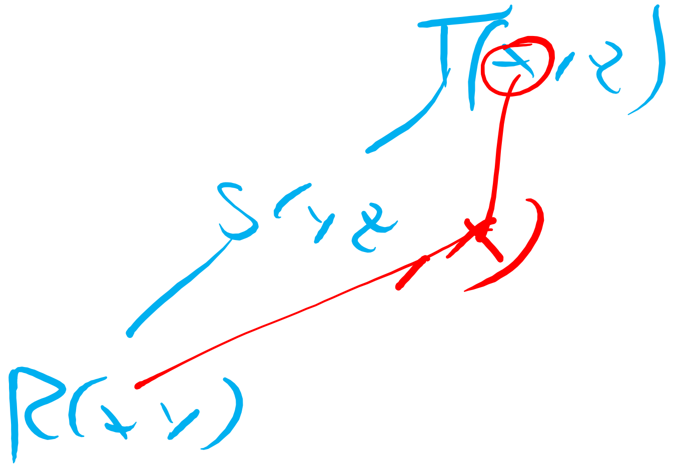
?

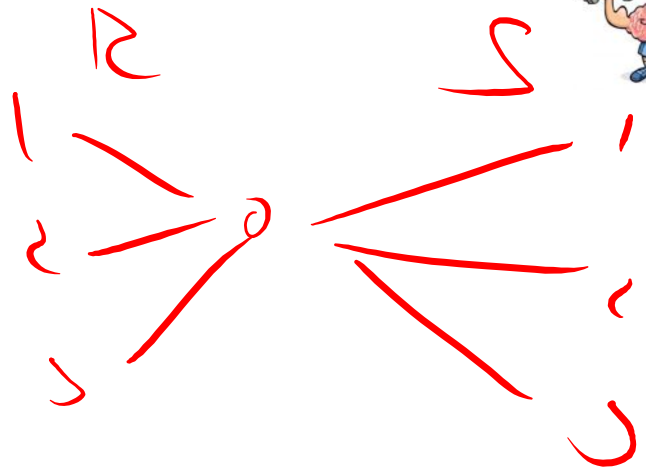
GYO reduction: Example 3: Triangle

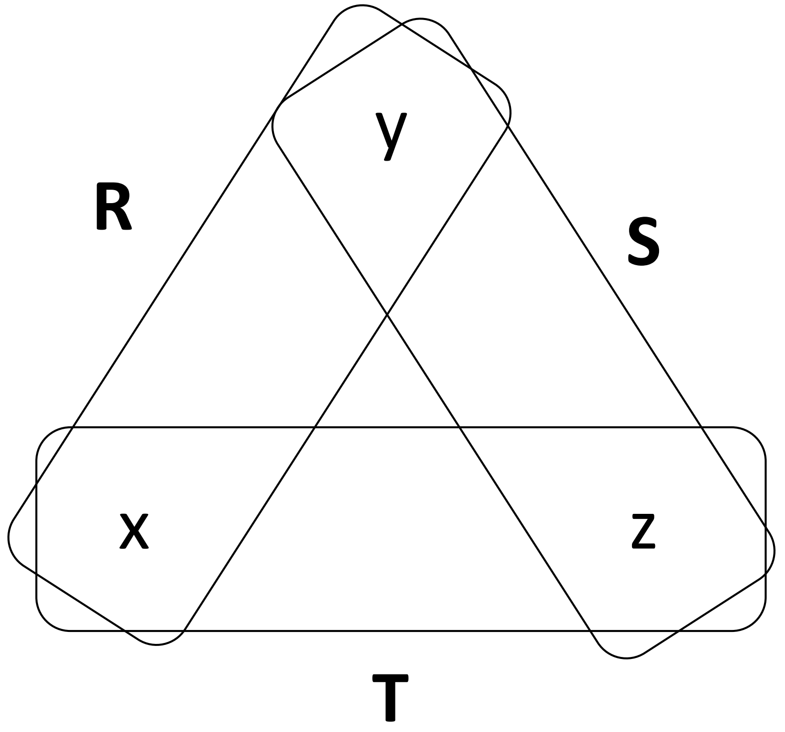
Q :- R(x,y), S(y,z), T(z,x).

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

- What about the triangle query?







GYO reduction: Example 4

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

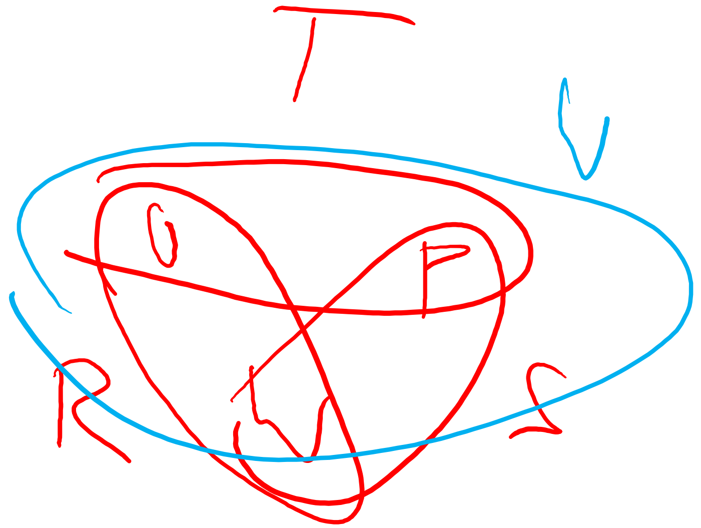
- GYO Ear removal
  - remove ears (= edges)
    - remove isolated nodes (variables)
    - remove consumed or empty edges (atoms)

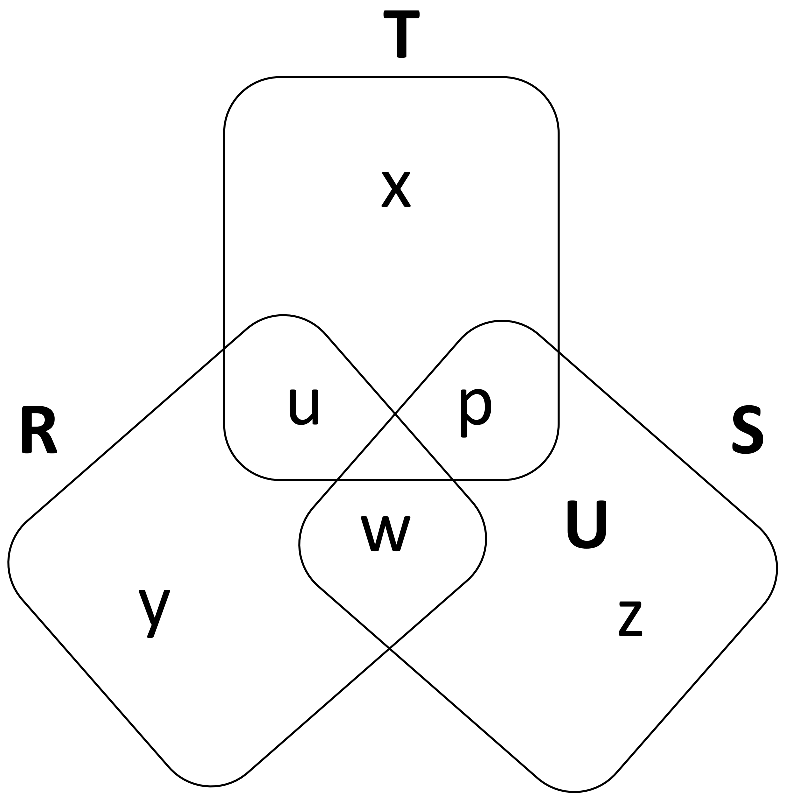
W(u,p,w)

$$\begin{tikzcd} & W & \\ R \arrow[ur] & S \arrow[u] & T \arrow[ul] \end{tikzcd}$$

?

Join tree



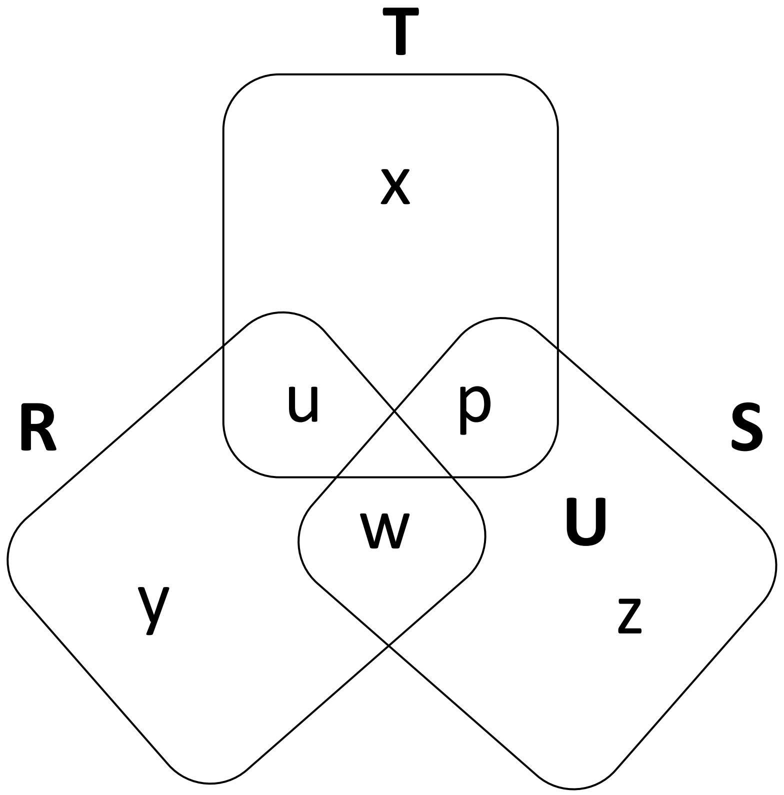


GYO reduction: Example 4

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

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

No Join tree, thus cyclic query!



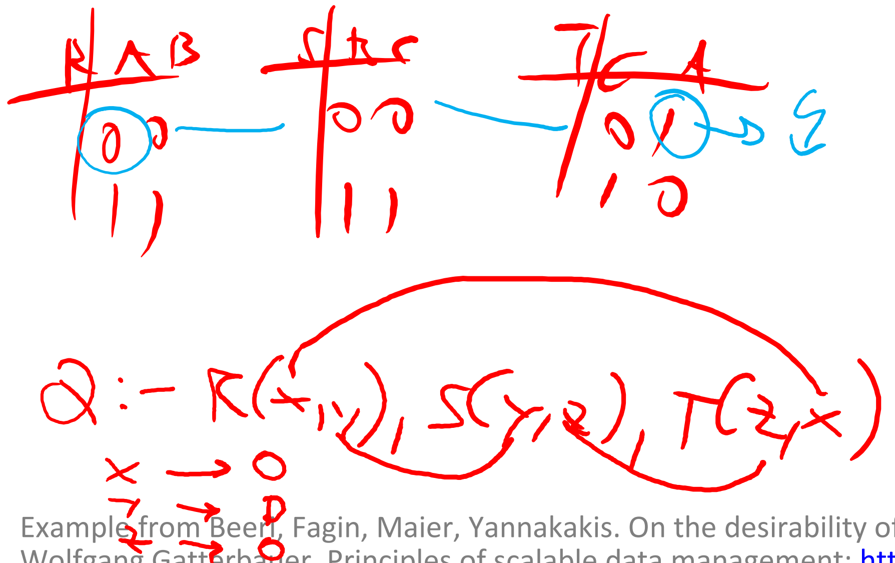
Acyclic queries

$(x \vee \bar{y} \vee z)$

c_1

c_1	x	y	z
	0	0	0
	0	0	1
	~~0~~	~~1~~	~~0~~
	0	1	1
	1	0	0
	1	0	1
	1	1	0
	1	1	1

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



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 $R \bowtie S \bowtie T$ = Ø, so the relations are not globally consistent

Reduction by Semi-joins

- Key insight for acyclic queries:
  - If there are no dangling tuples, then <u>the result can never shrink with an additional join</u>
  - 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:

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

  - 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: $R(A_1,\ldots,A_n)$, $S(B_1,\ldots,B_m)$, Output: $T(A_1,\ldots,A_n)$

R(x,y,z), S(y,z,n)

- The law of semijoins is:

$$\boxed{R \bowtie S = (R \ltimes_C S) \bowtie S}$$

  - 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(|Input|+|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)

x	y
a_1	b_1
a_1	b_2
a_1	b_3
a_2	b_1
a_2	b_2
a_2	b_3
a_3	b_1
a_3	b_2
a_3	b_3

S(y,z)

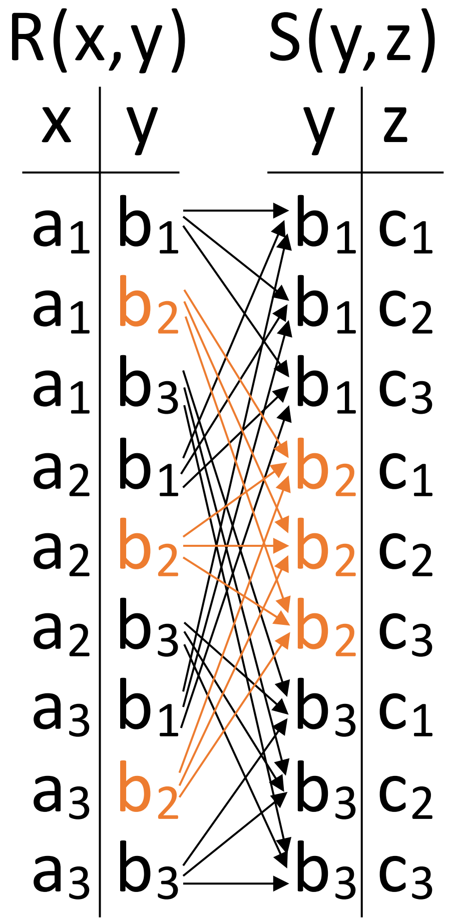
y	z
b_1	c_1
b_1	c_2
b_1	c_3
b_2	c_1
b_2	c_2
b_2	c_3
b_3	c_1
b_3	c_2
b_3	c_3

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)

x	y
a_1	b_1
a_1	b_2
a_1	b_3
a_2	b_1
a_2	b_2
a_2	b_3
a_3	b_1
a_3	b_2
a_3	b_3

S(y,z)

y	z
b_1	c_1
b_1	c_2
b_1	c_3
b_2	c_1
b_2	c_2
b_2	c_3
b_3	c_1
b_3	c_2
b_3	c_3

How many joins are between R and S?

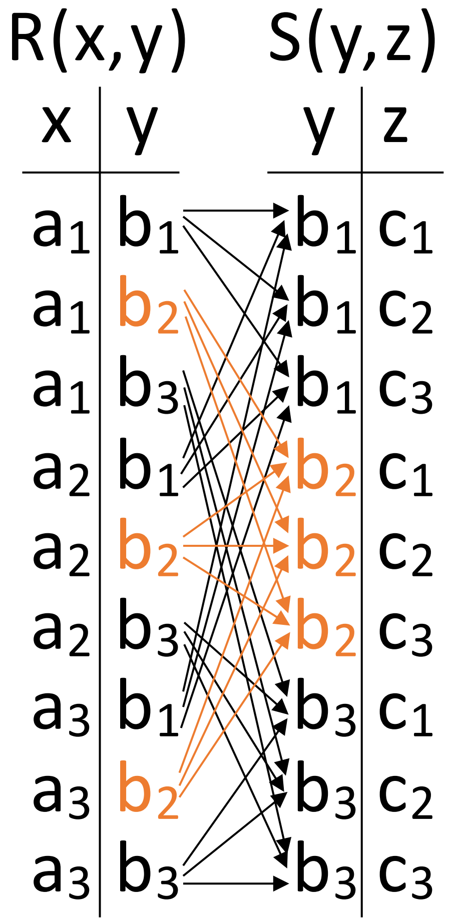
27

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)

x	y
a_1	b_1
a_1	b_2
a_1	b_3
a_2	b_1
a_2	b_2
a_2	b_3
a_3	b_1
a_3	b_2
a_3	b_3

S(y,z)

y	z
b_1	c_1
b_1	c_2
b_1	c_3
b_2	c_1
b_2	c_2
b_2	c_3
b_3	c_1
b_3	c_2
b_3	c_3

How many joins are between R and S?

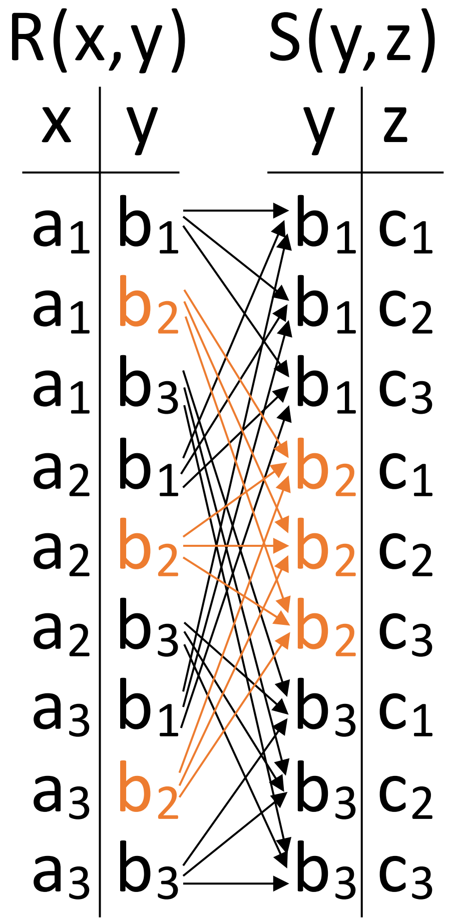
27

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

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



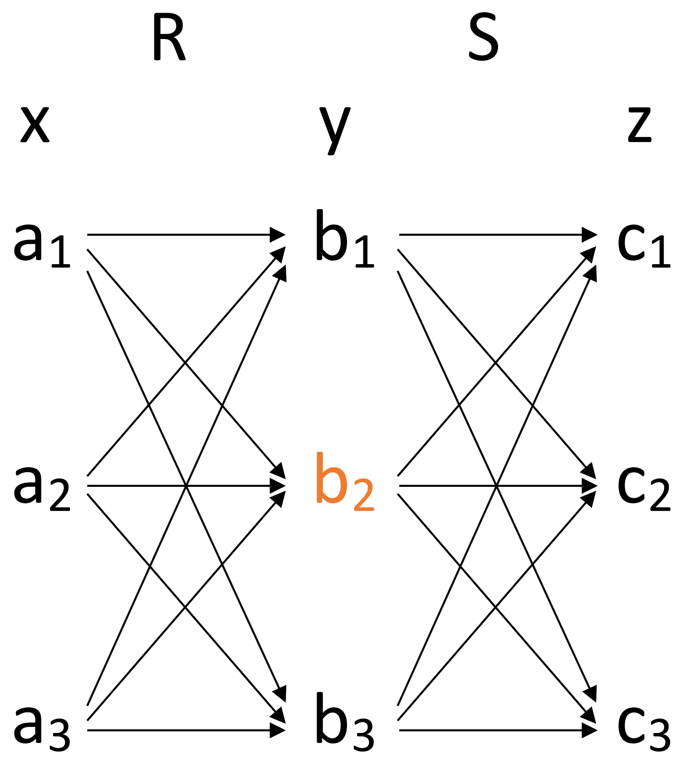
R(x,y)

x	y
a_1	b_1
a_1	b_2
a_1	b_3
a_2	b_1
a_2	b_2
a_2	b_3
a_3	b_1
a_3	b_2
a_3	b_3

S(y,z)

y	z
b_1	c_1
b_1	c_2
b_1	c_3
b_2	c_1
b_2	c_2
b_2	c_3
b_3	c_1
b_3	c_2
b_3	c_3

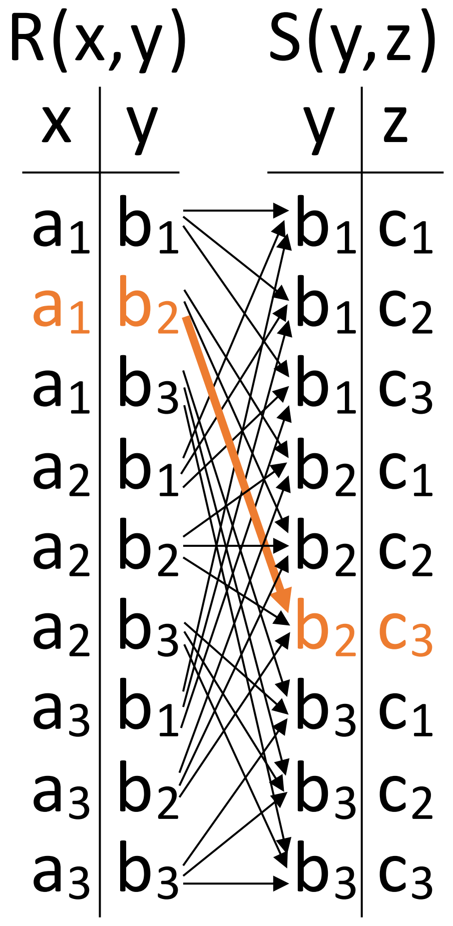
- **Nodes are the tuples**
- Edges are the joins



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

Application

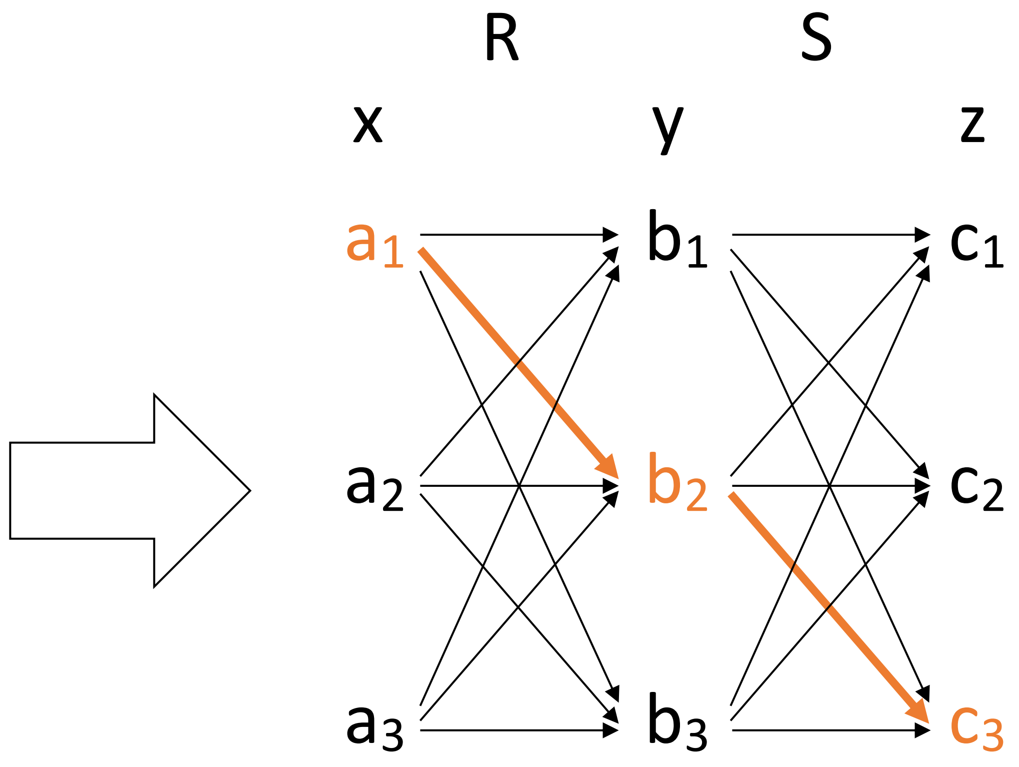
Q(x,y,z,u,v,w) :- R(x,y),S(y,z),T(z,u),K(u,v),L(v,w).



x	y
a_1	b_1
a_1	b_2
a_1	b_3
a_2	b_1
a_2	b_2
a_2	b_3
a_3	b_1
a_3	b_2
a_3	b_3

y	z
b_1	c_1
b_1	c_2
b_1	c_3
b_2	c_1
b_2	c_2
b_2	c_3
b_3	c_1
b_3	c_2
b_3	c_3

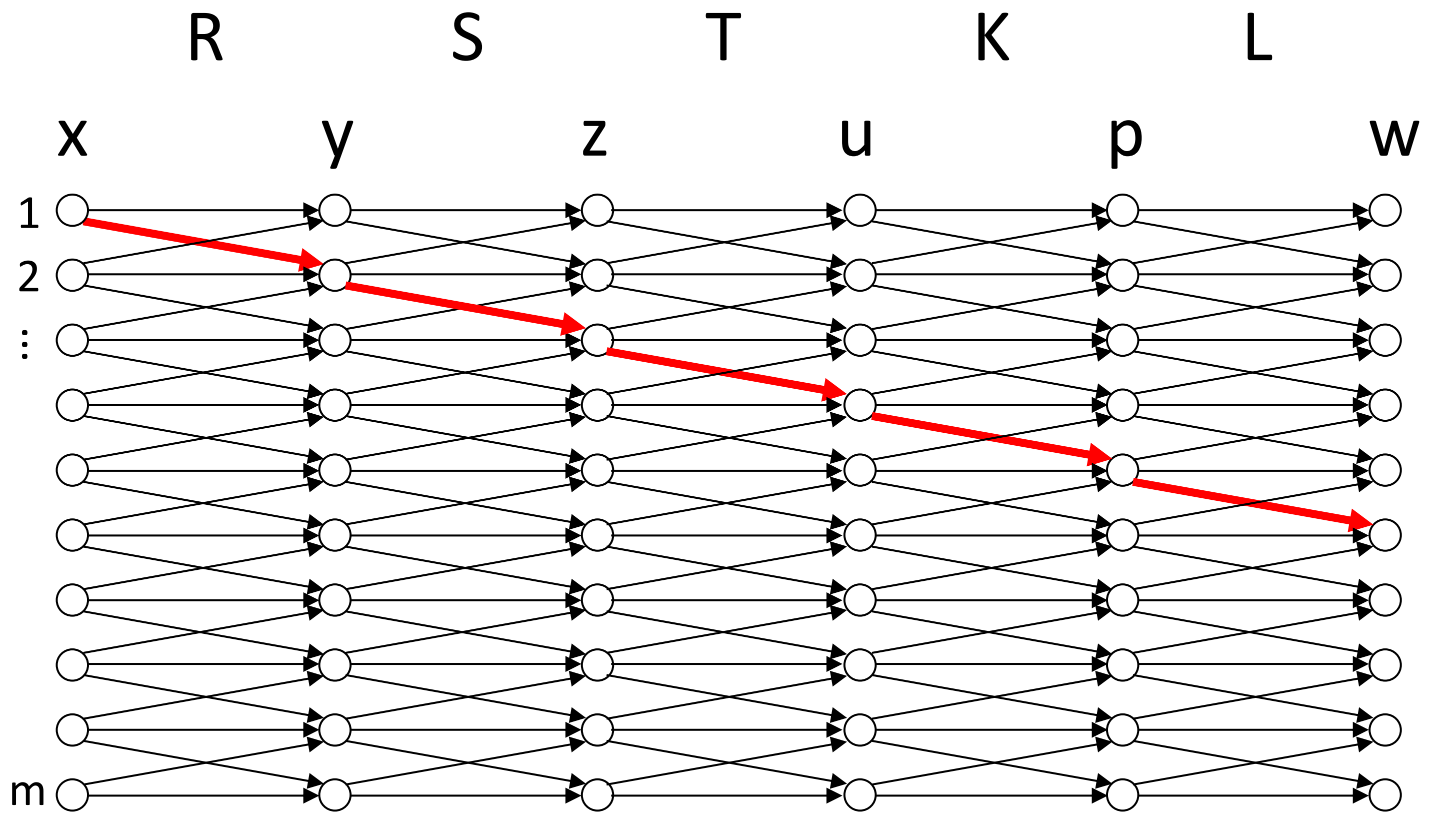
- **Nodes are the tuples**
- Edges are the joins



- **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).



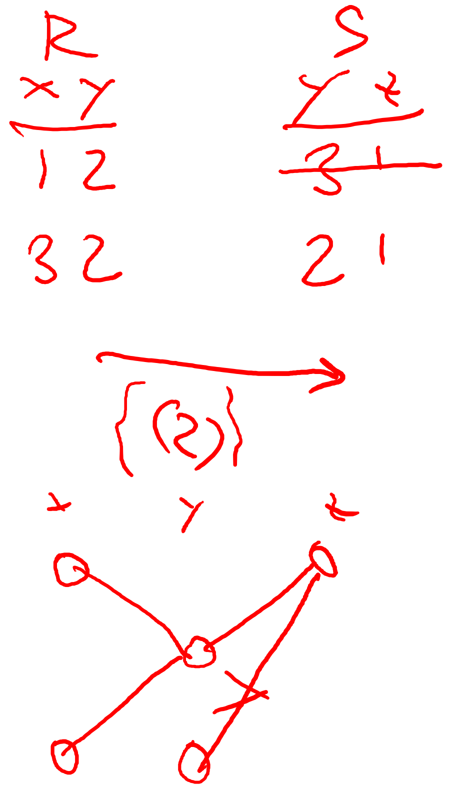
- Intermediate relations of size up to m^5 (m is here domain size!)
- But final answer can be as small as 0 (or 1...)

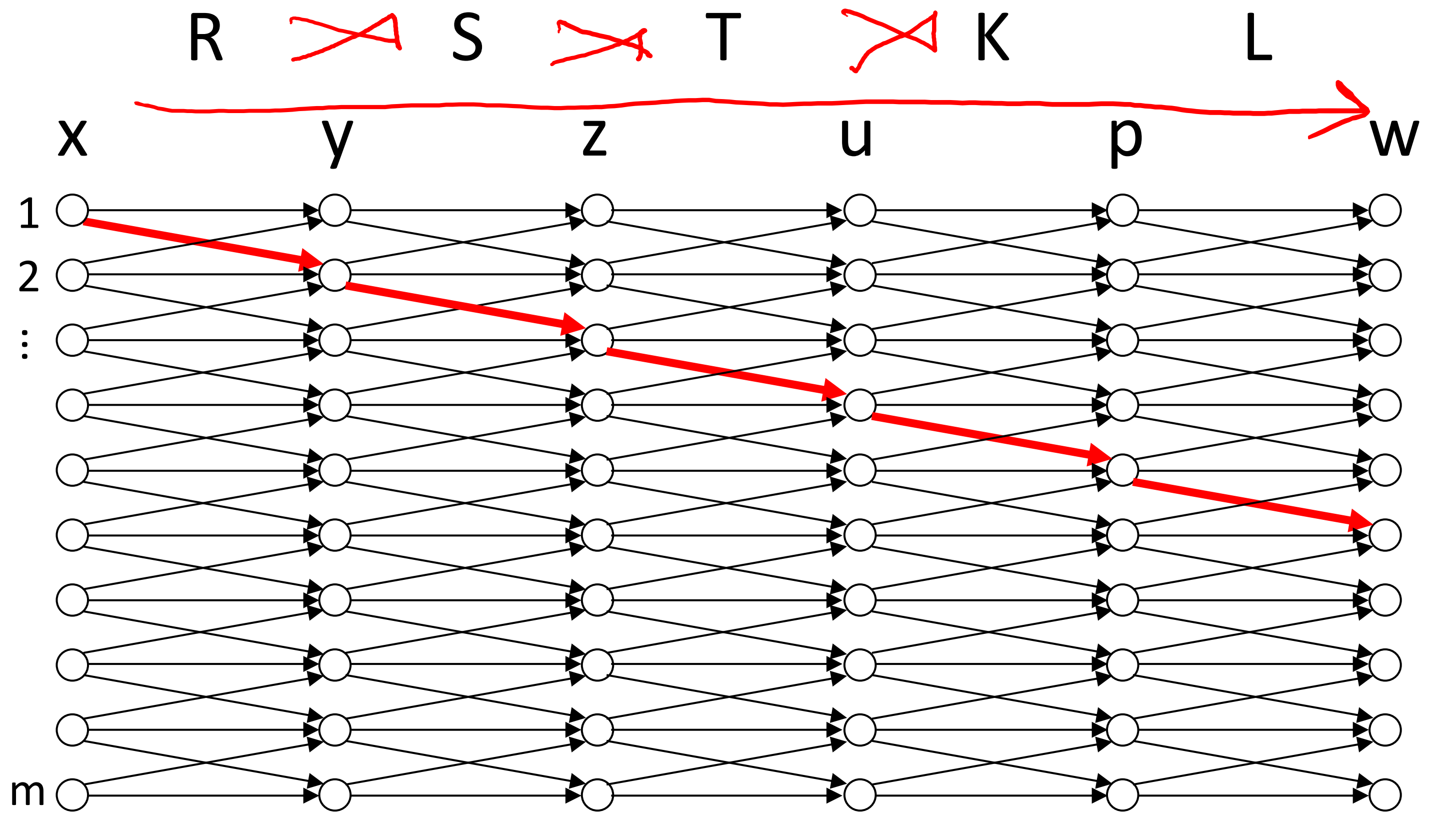
Application

$\pi_y = \pi_x$

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)

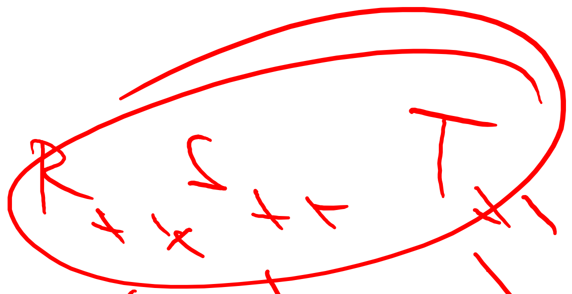


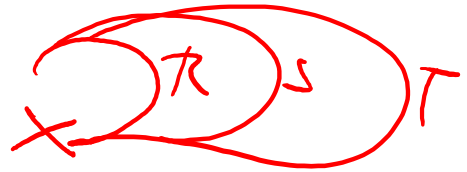


S := S ⋉ R
T := T ⋉ S
K := K ⋉ T
L := L ⋉ K
FWD
K := K ⋉ L
T := T ⋉ K
S := S ⋉ T
R := R ⋉ S
BACK

- 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

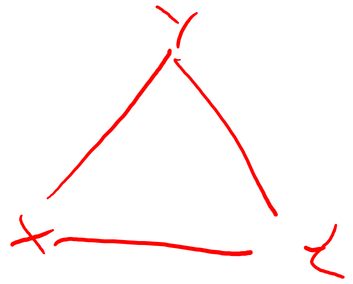


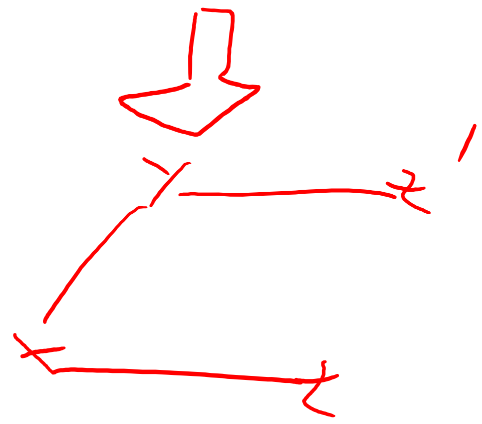


Q^{Δ} :- $R(x,y), S(y,z), T(z,x)$

Q^{P4} :- $R(x,y), S(y,z), T(z',x), (z = z')$

$Q^{P4} \supseteq Q^{\Delta}$





$z = z'$

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

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

one of several possible ones

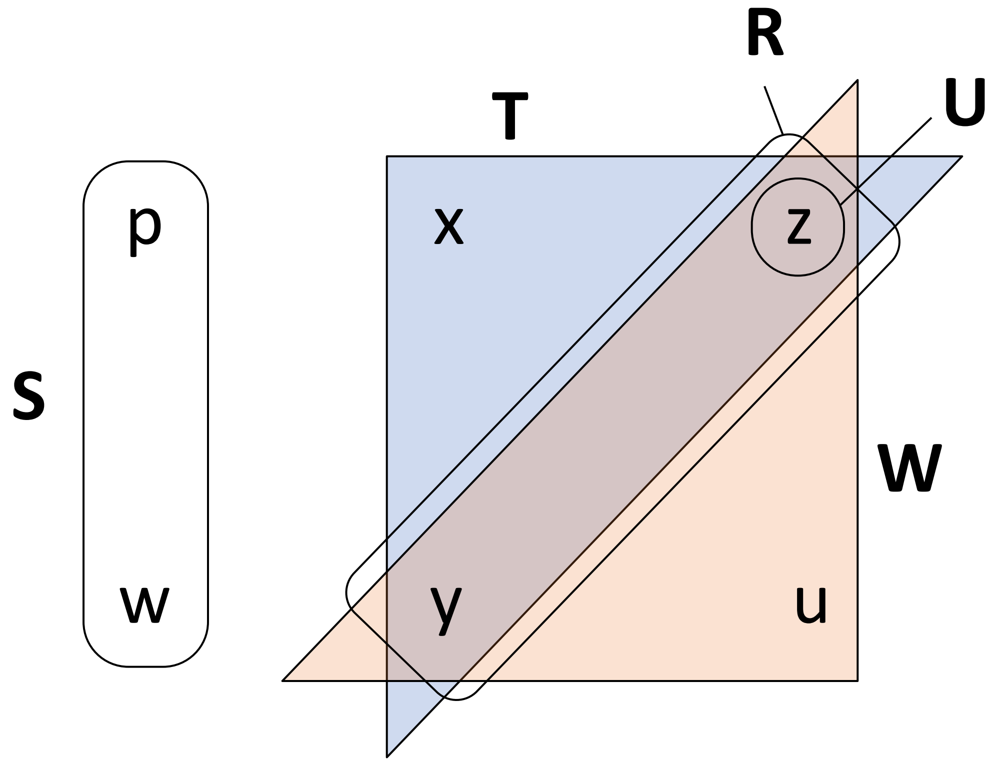
Rooted Join tree

?

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).

Hypergraph



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

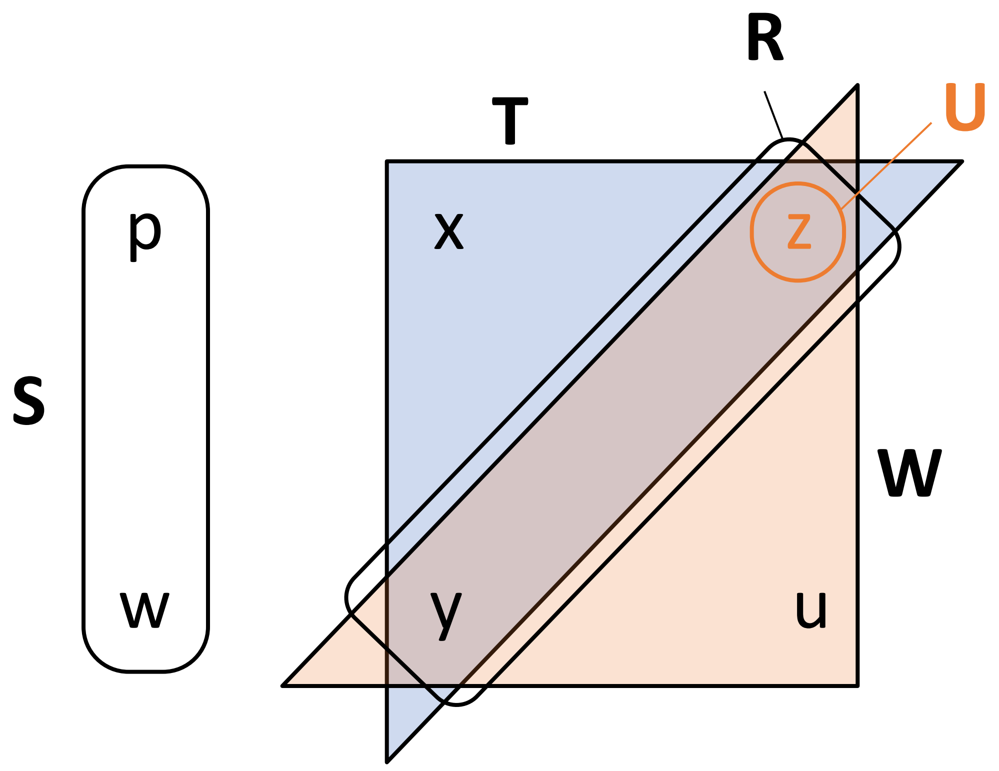
Rooted Join tree

?

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).

Hypergraph



Rooted Join tree

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

z = -Ø

U(z)

z
c_1
c_2
c_3

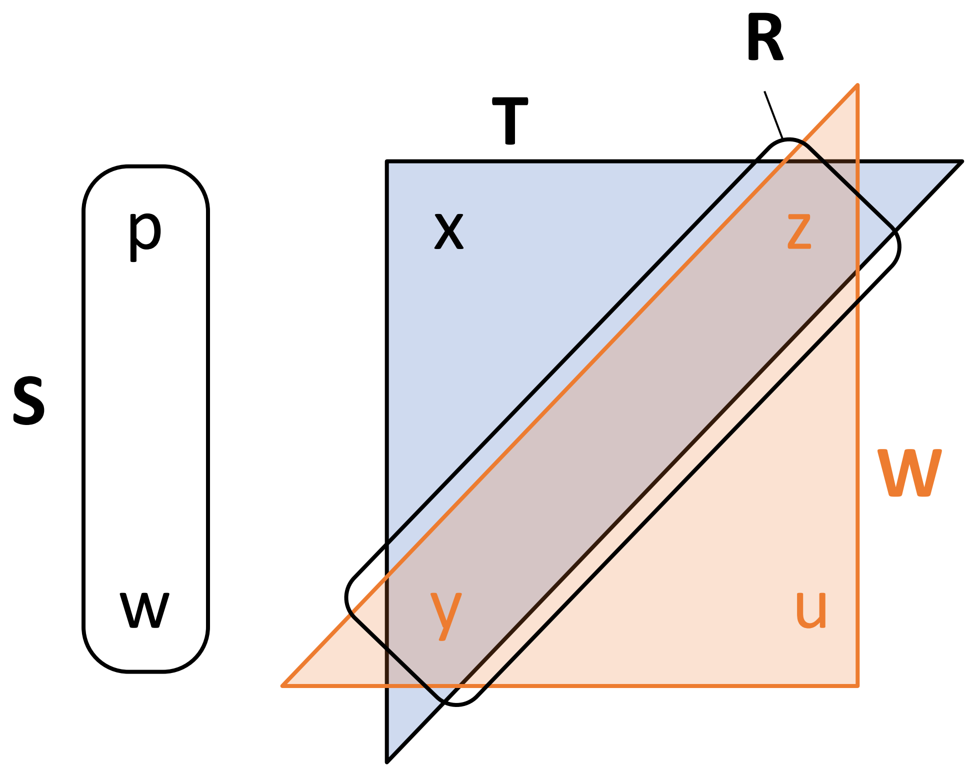
W(y,z,u)

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

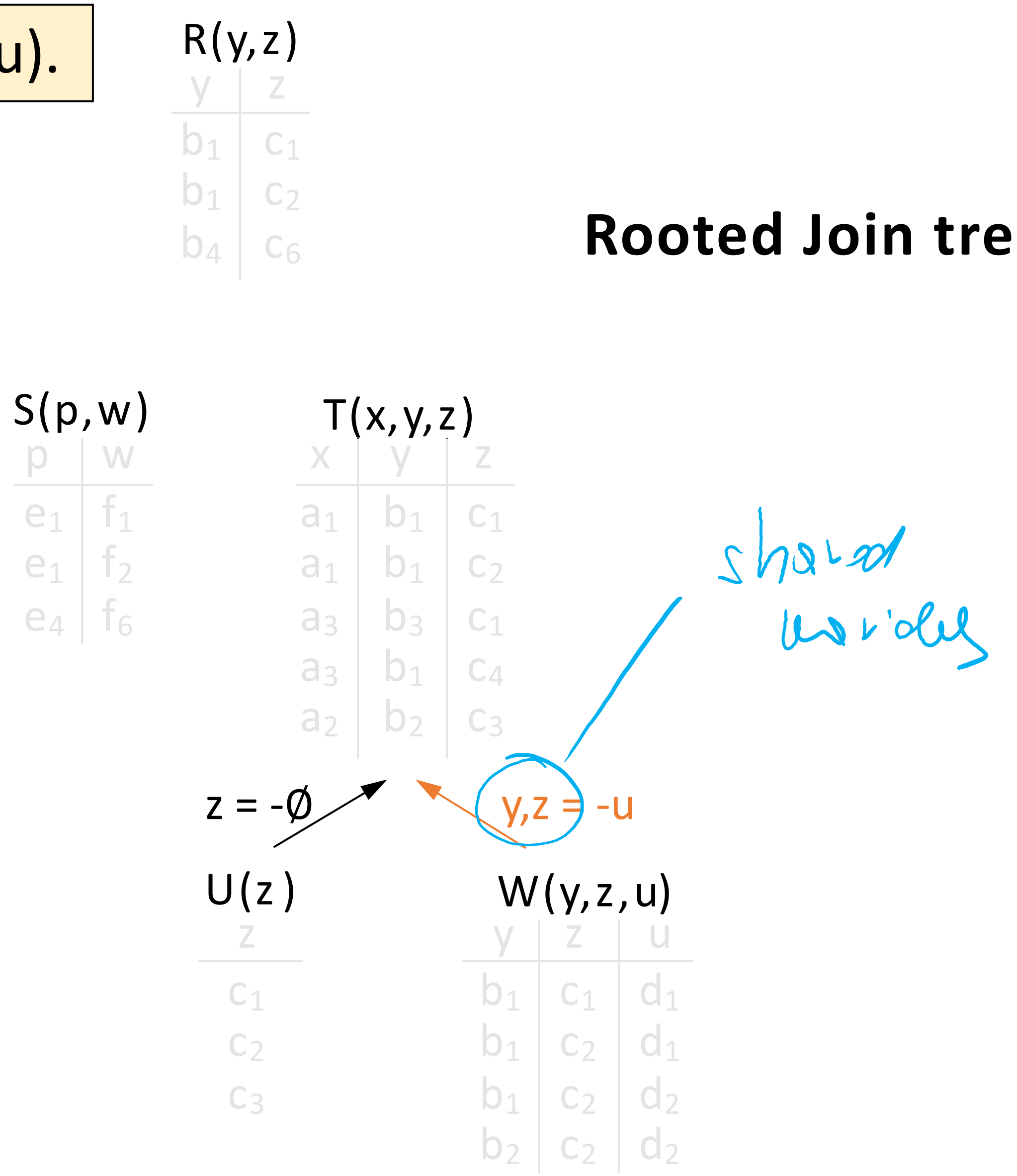
Yannakakis Algorithm example: start from join tree

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

Hypergraph



Rooted Join tree



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

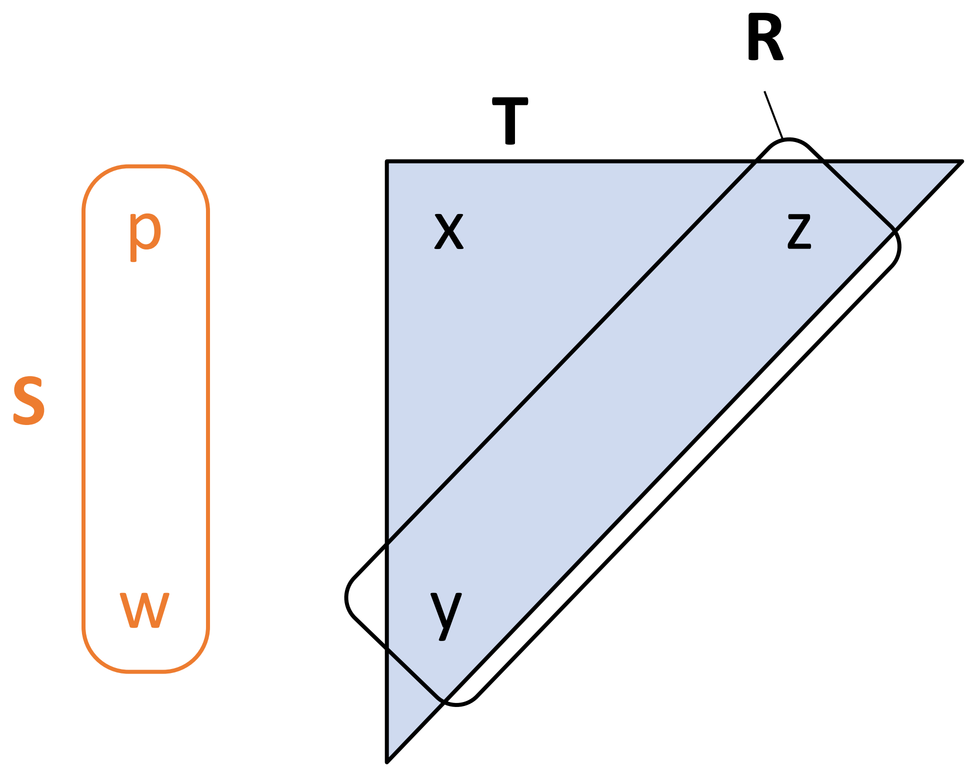
W(y,z,u)

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

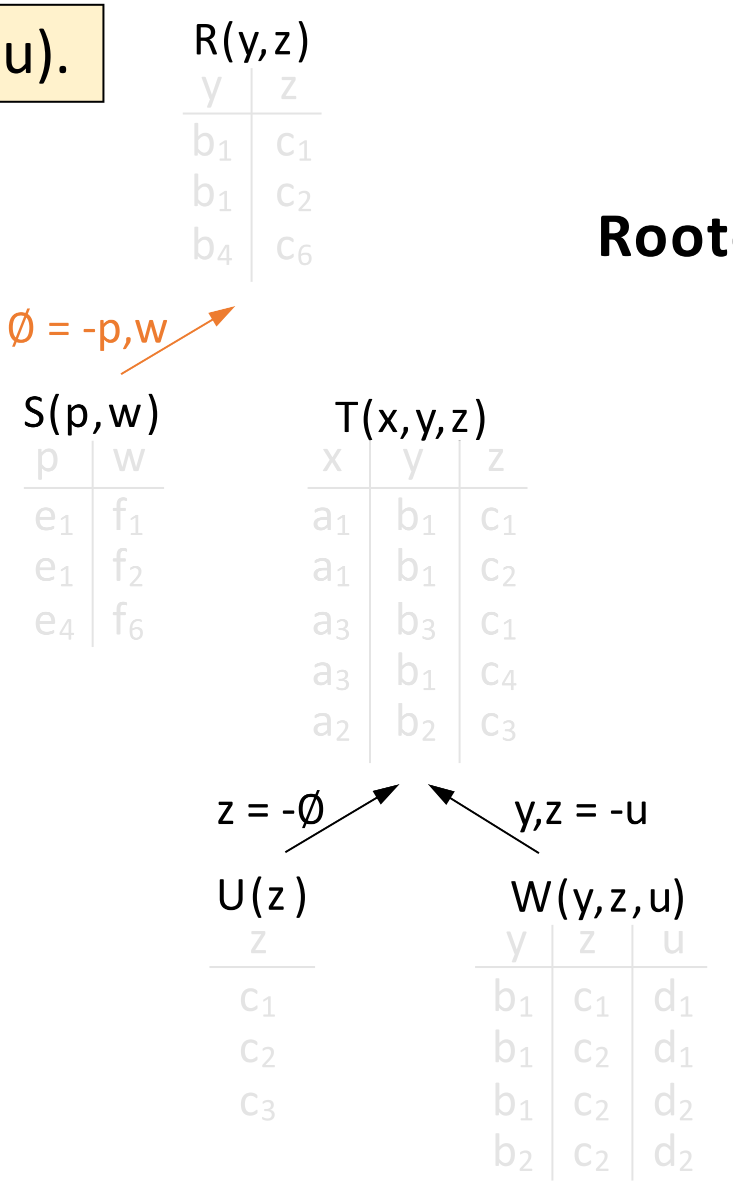
Yannakakis Algorithm example: start from join tree

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

Hypergraph



Rooted Join tree



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

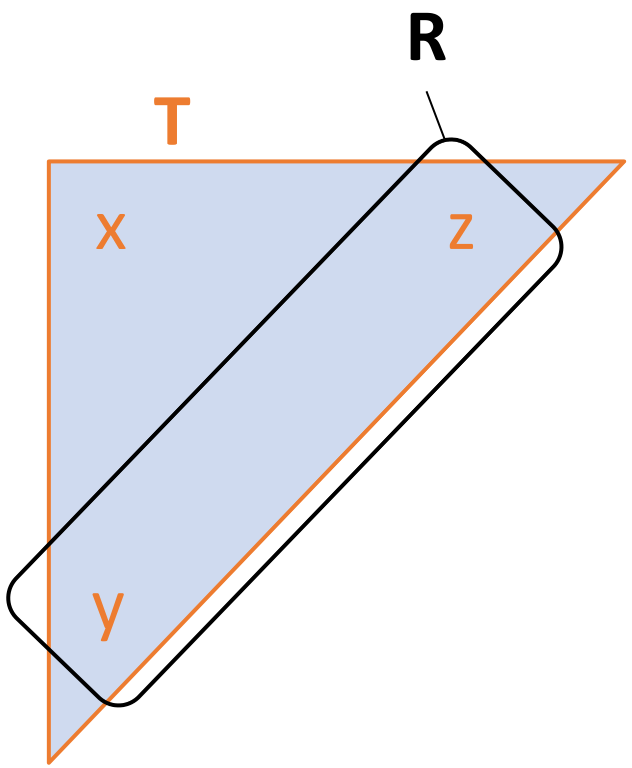
W(y,z,u)

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

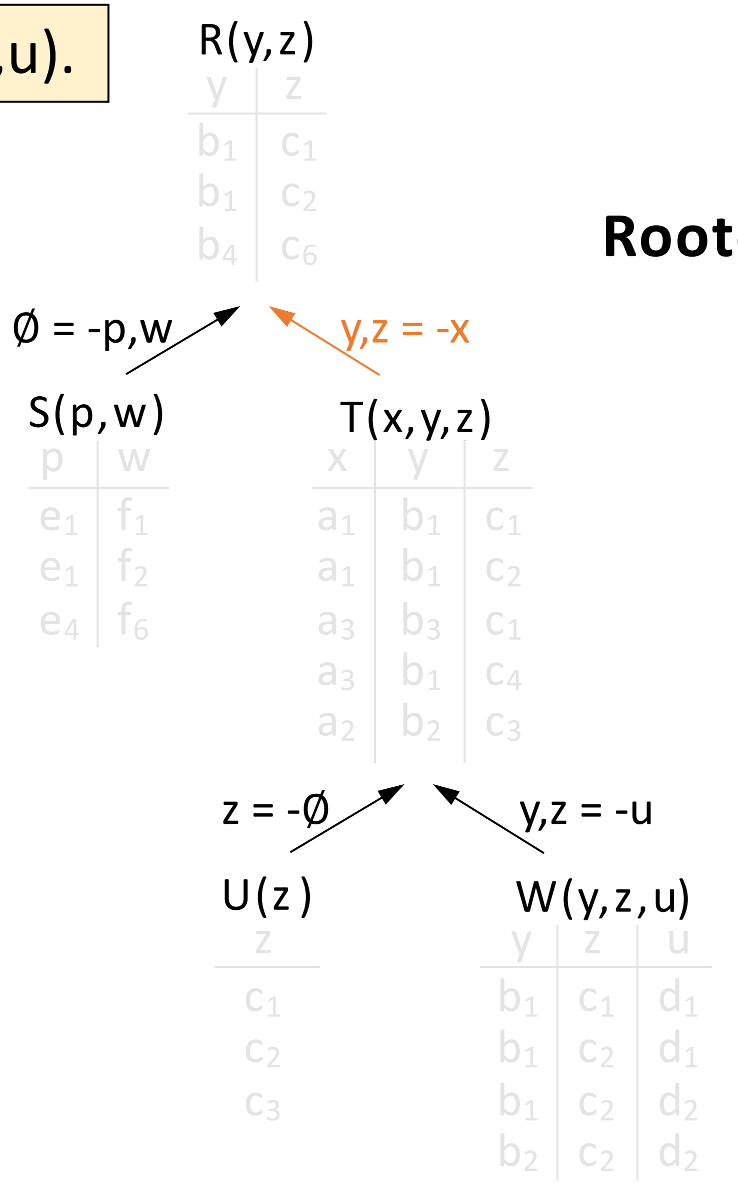
Yannakakis Algorithm example: start from join tree

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

Hypergraph



Rooted Join tree



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

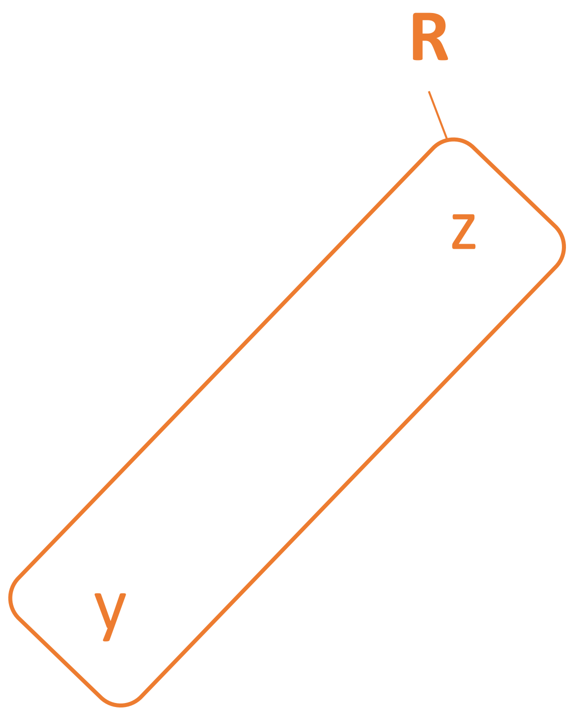
W(y,z,u)

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

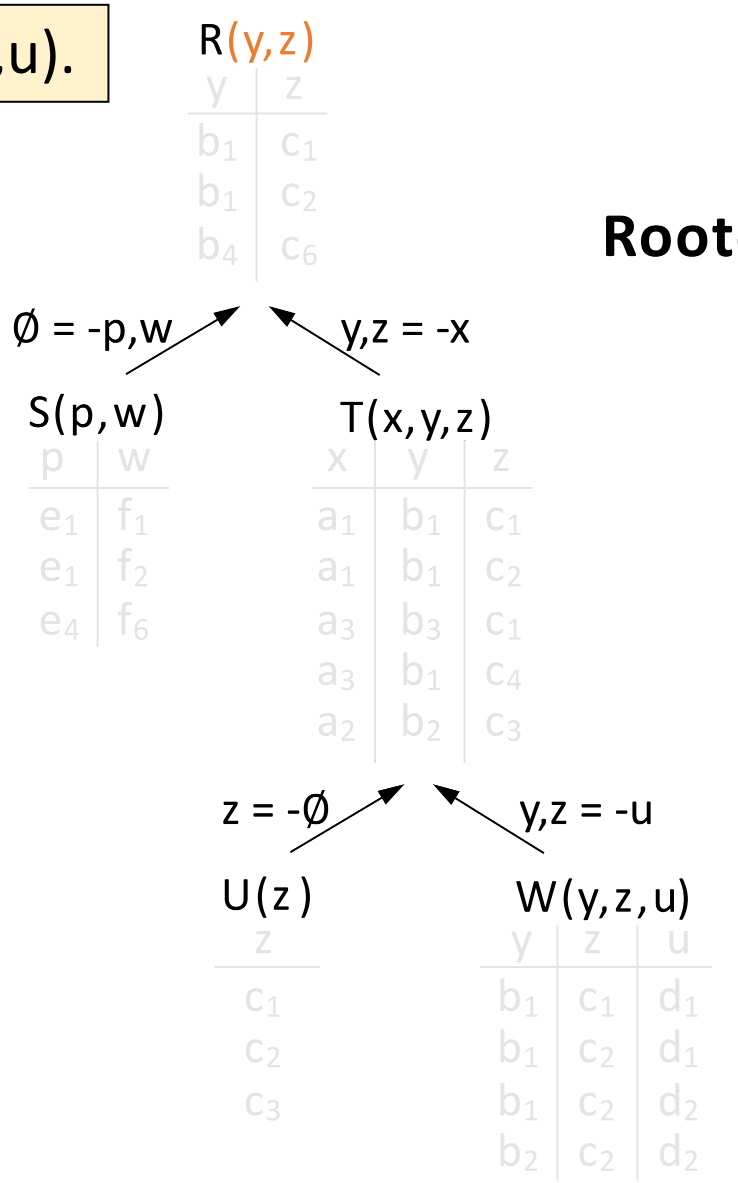
Yannakakis Algorithm example: start from join tree

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

Hypergraph



Rooted Join tree



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

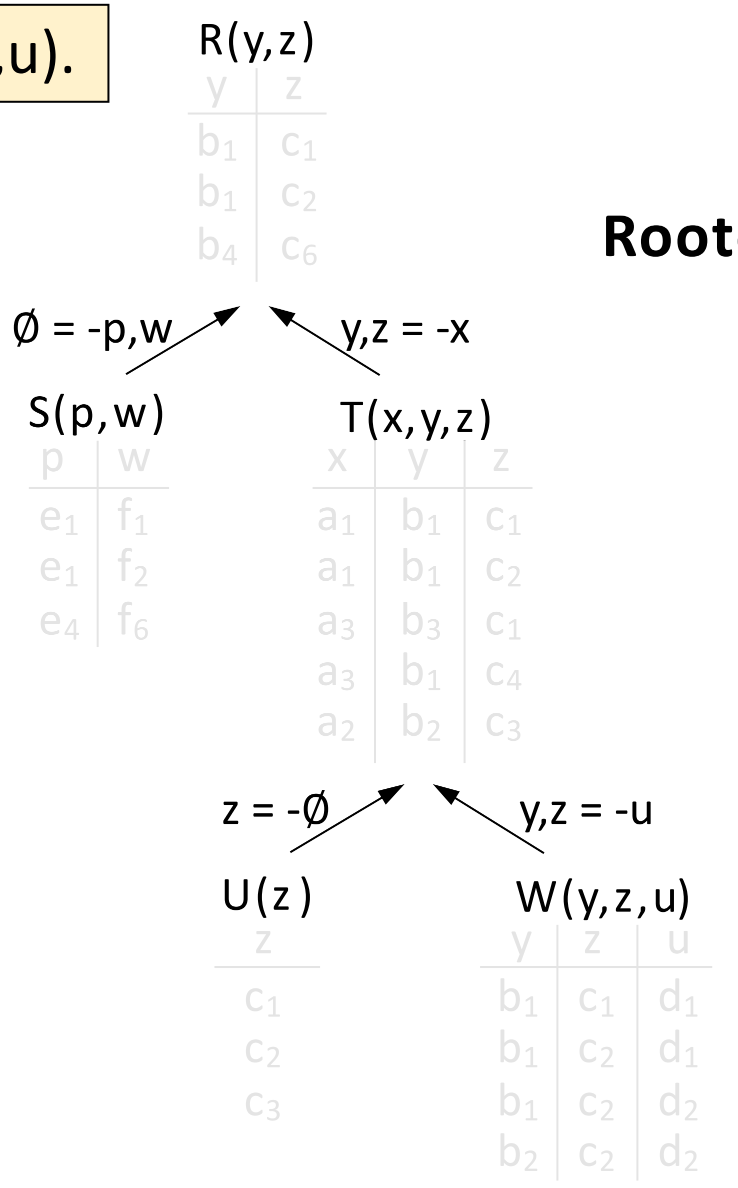
W(y,z,u)

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: 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).

Rooted Join tree



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)

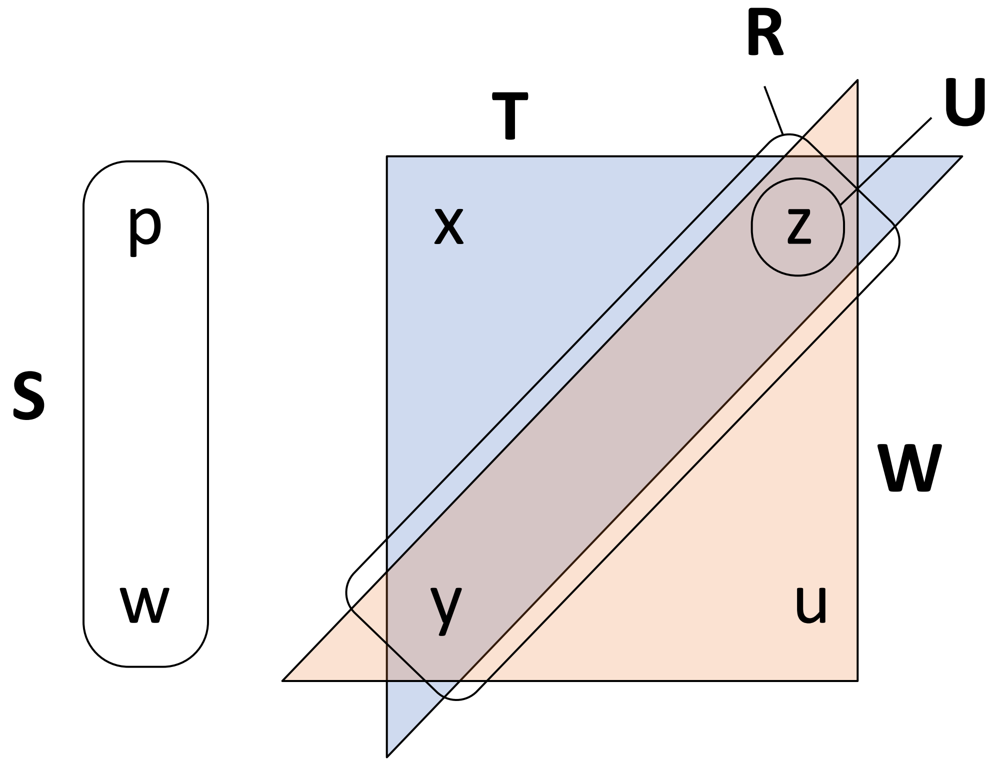
z
c_1
c_2
c_3

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: 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).



Rooted Join tree

$$\begin{tikzcd}
 & R(y,z) & & & \\
S(p,w) \arrow[ur, "\emptyset = -p{,}w"] & & & T(x,y,z) \arrow[ull, "y{,}z = -x"'] & \\
 & & U(z) \arrow[ur, "z = -\emptyset"] & & W(y,z,u) \arrow[ul, "y{,}z = -u"']
\end{tikzcd}$$

shared variables

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)

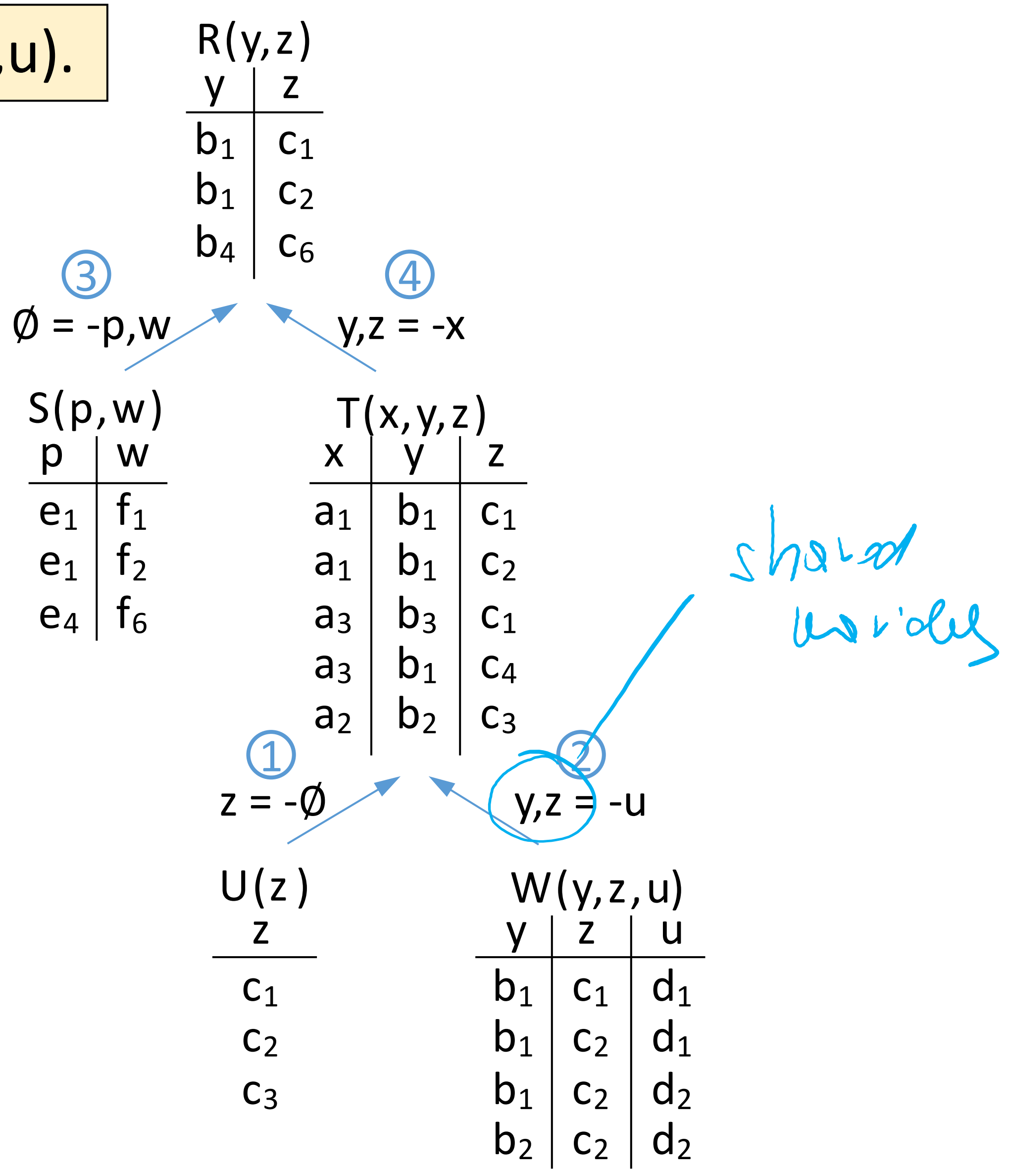
y	z	u
b_1	c_1	d_1
b_1	c_2	d_1
b_1	c_2	d_2
b_2	c_2	d_2

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
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)

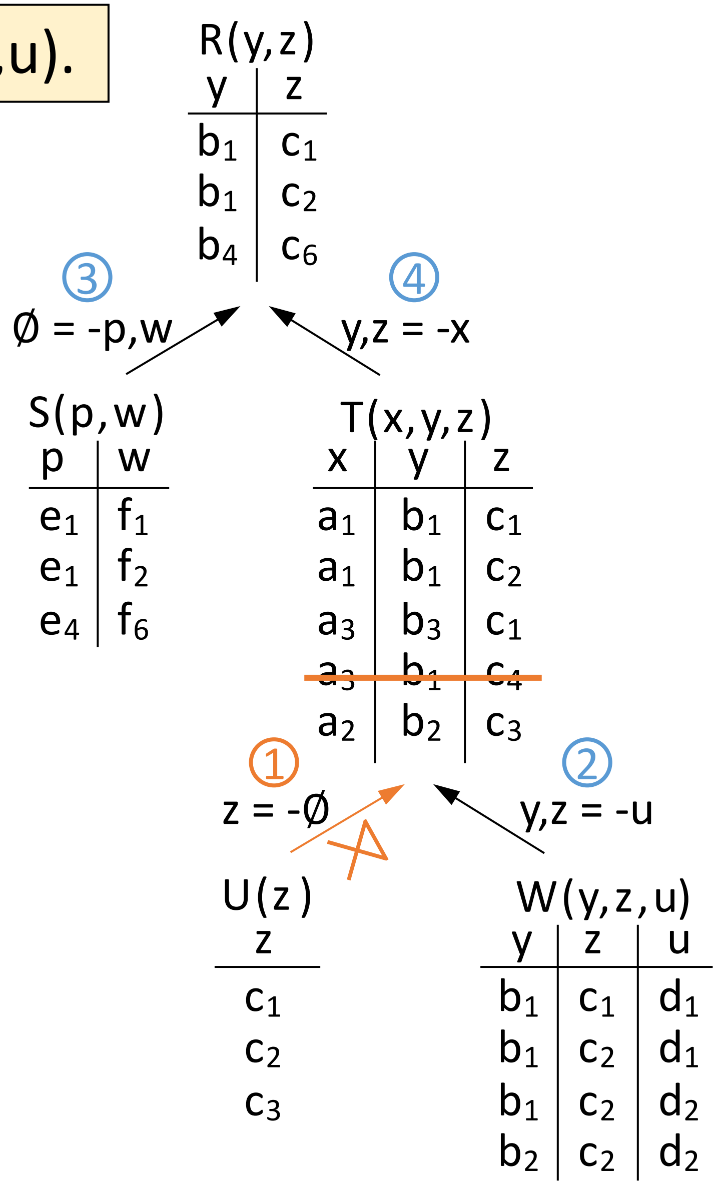
y	z	u
b_1	c_1	d_1
b_1	c_2	d_1
b_1	c_2	d_2
b_2	c_2	d_2

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
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)

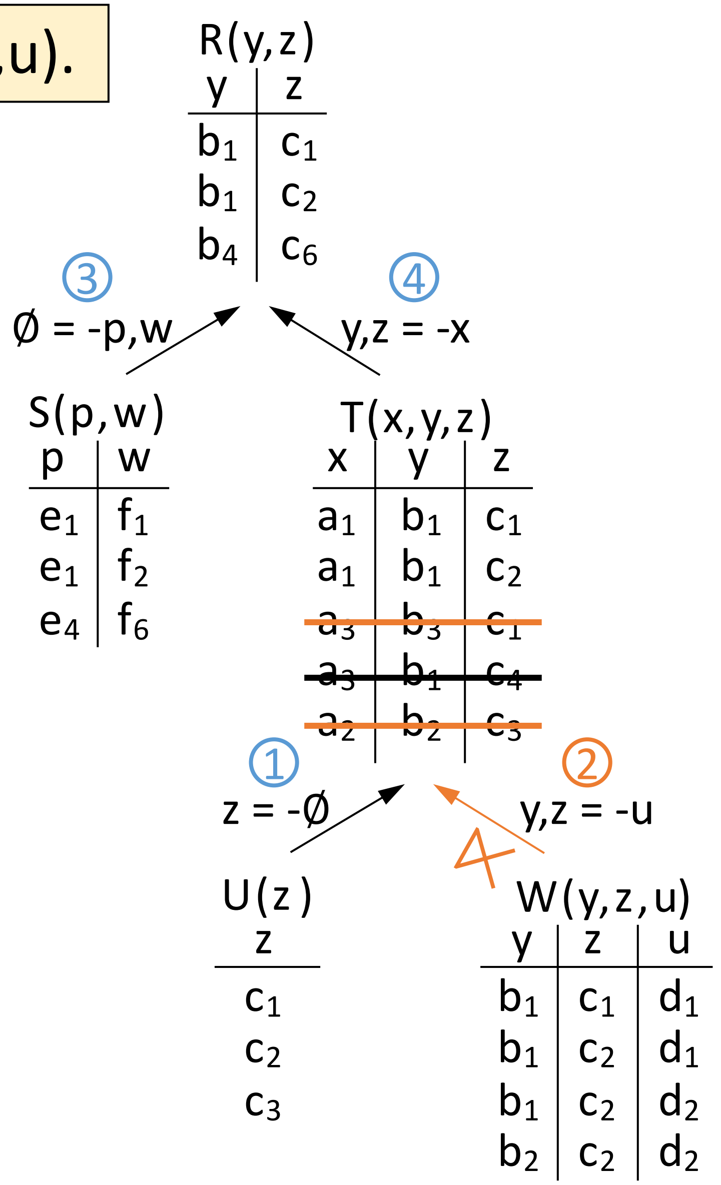
y	z	u
b_1	c_1	d_1
b_1	c_2	d_1
b_1	c_2	d_2
b_2	c_2	d_2

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
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)

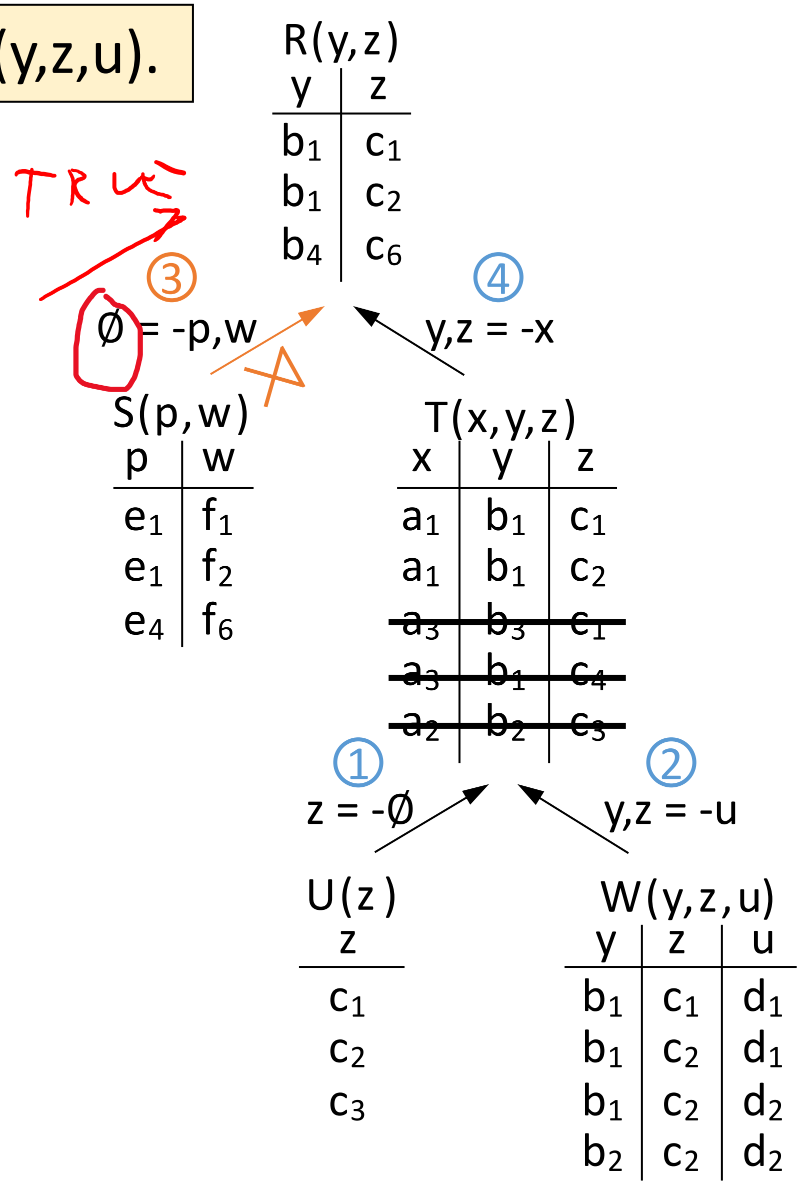
y	z	u
b_1	c_1	d_1
b_1	c_2	d_1
b_1	c_2	d_2
b_2	c_2	d_2

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
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)

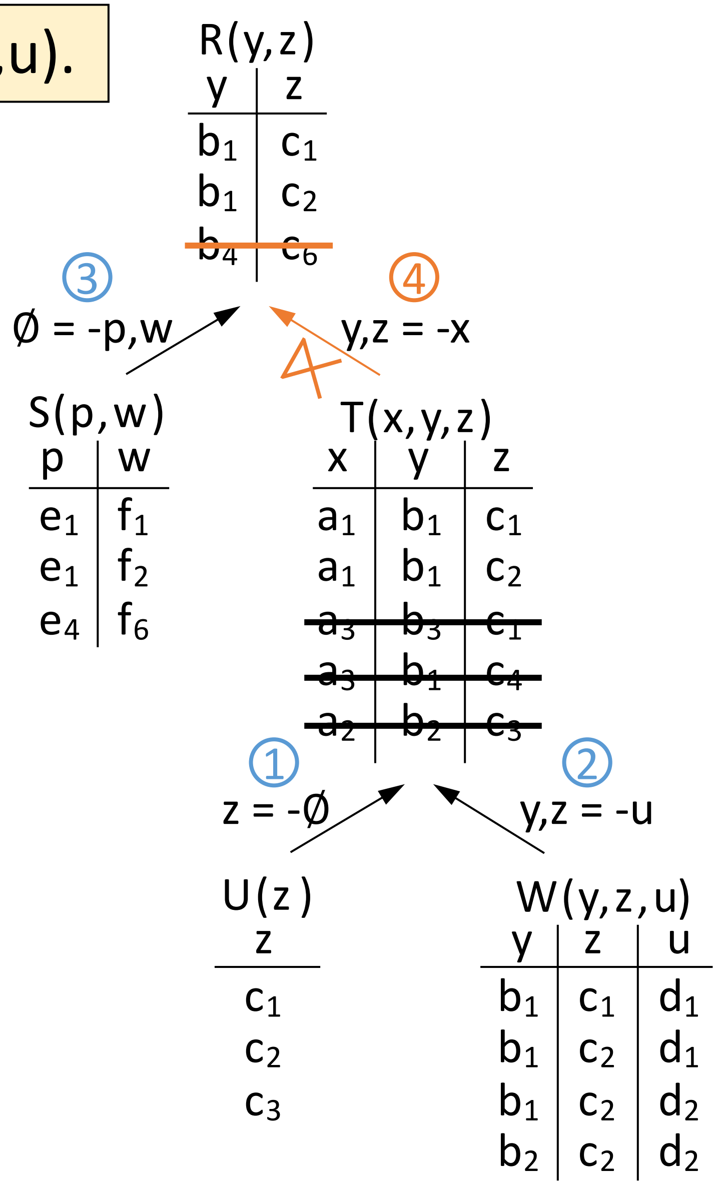
y	z	u
b_1	c_1	d_1
b_1	c_2	d_1
b_1	c_2	d_2
b_2	c_2	d_2

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
~~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)

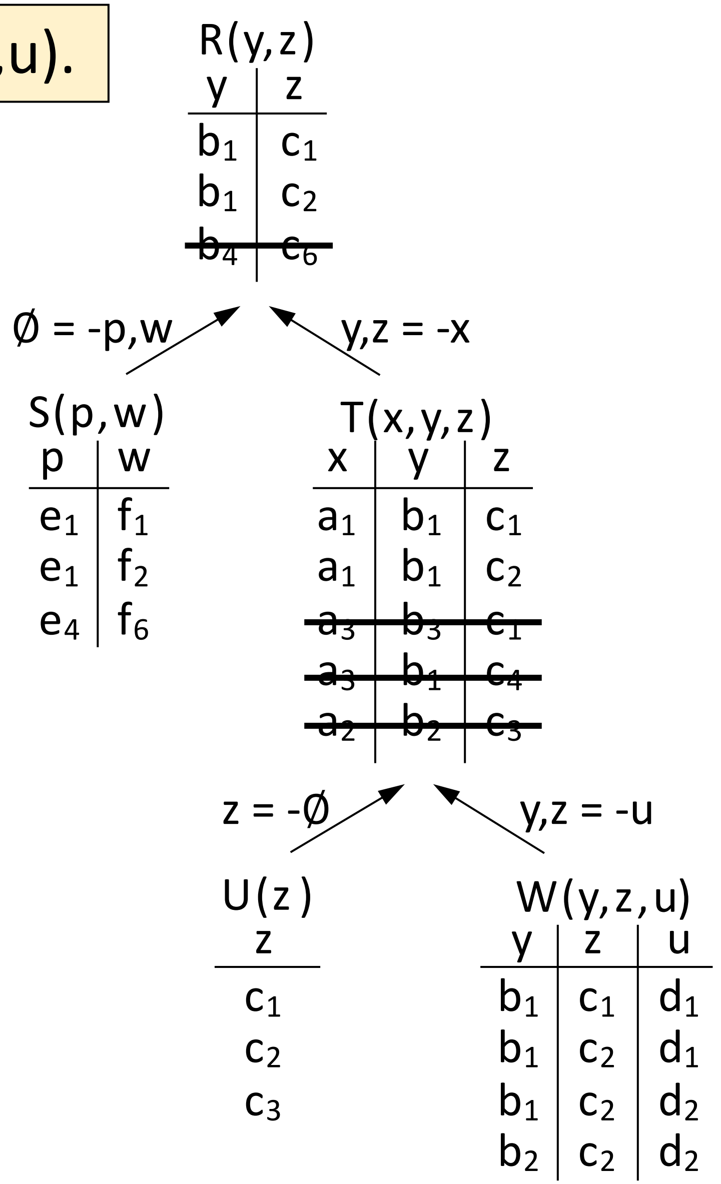
y	z	u
b_1	c_1	d_1
b_1	c_2	d_1
b_1	c_2	d_2
b_2	c_2	d_2

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
~~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

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.

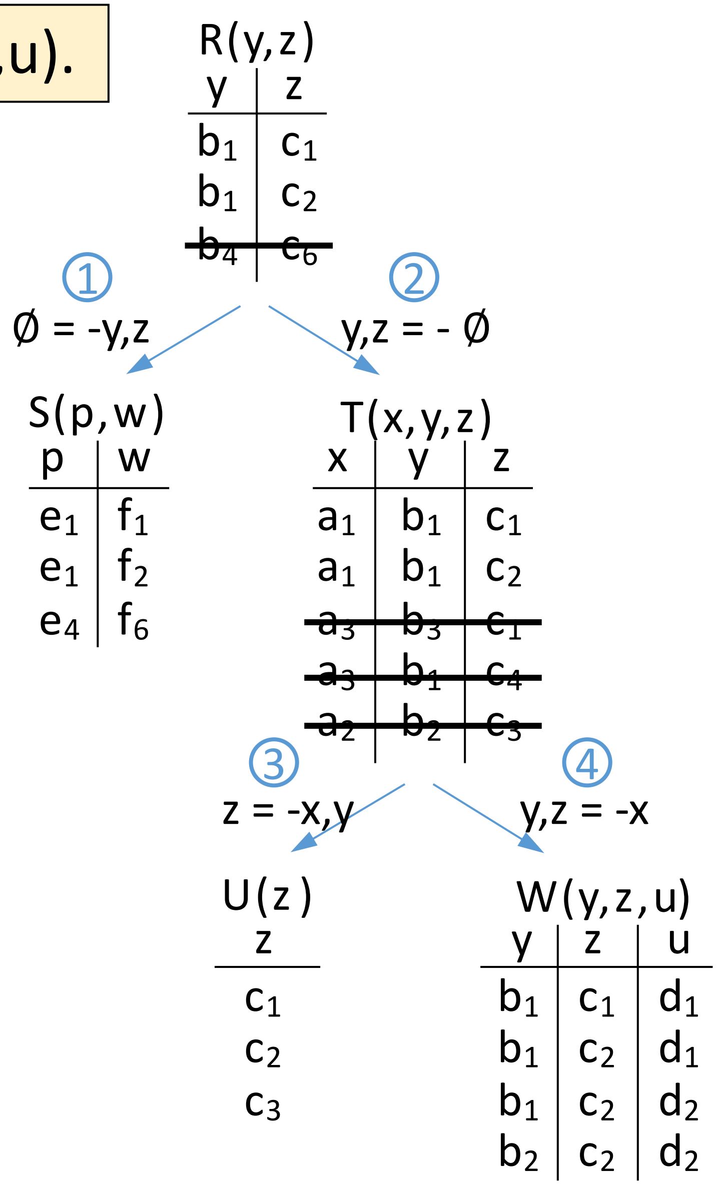
In other words, with a sequence of only local updates, we have aggregated at the root all necessary information to answer the Boolean query.

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 ⋉ (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**



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)

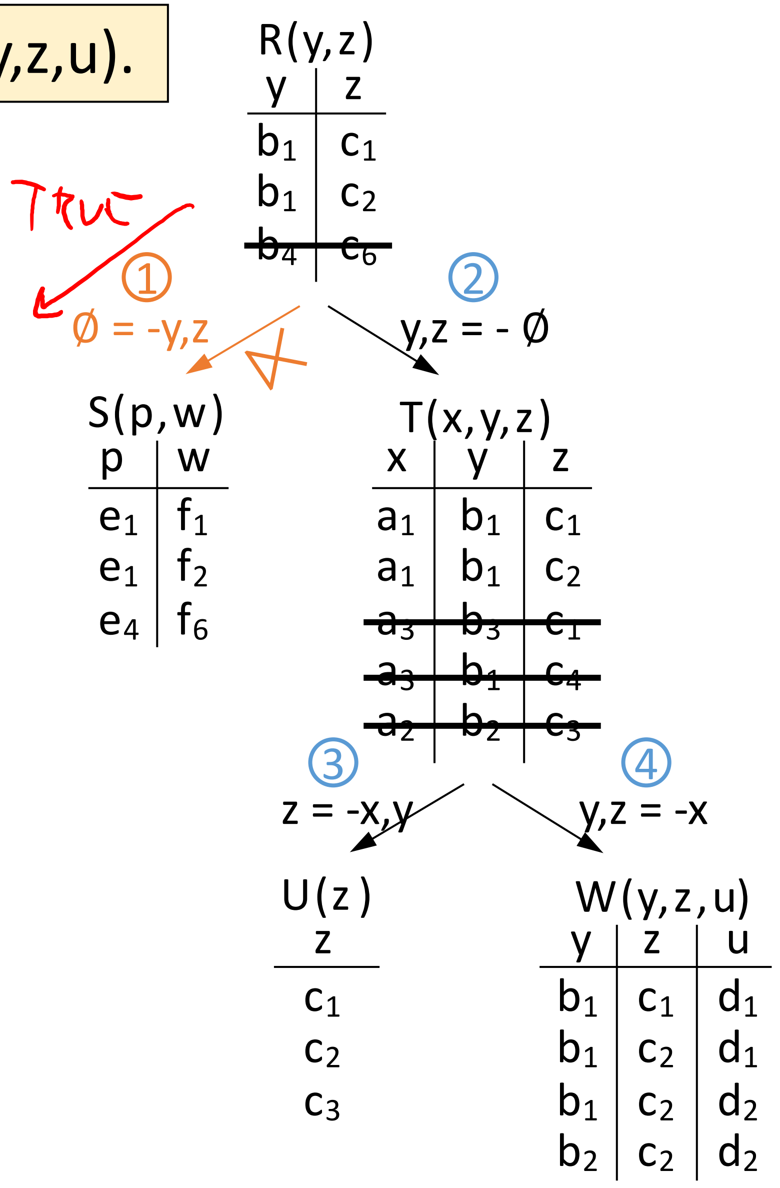
y	z	u
b_1	c_1	d_1
b_1	c_2	d_1
b_1	c_2	d_2
b_2	c_2	d_2

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**



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)

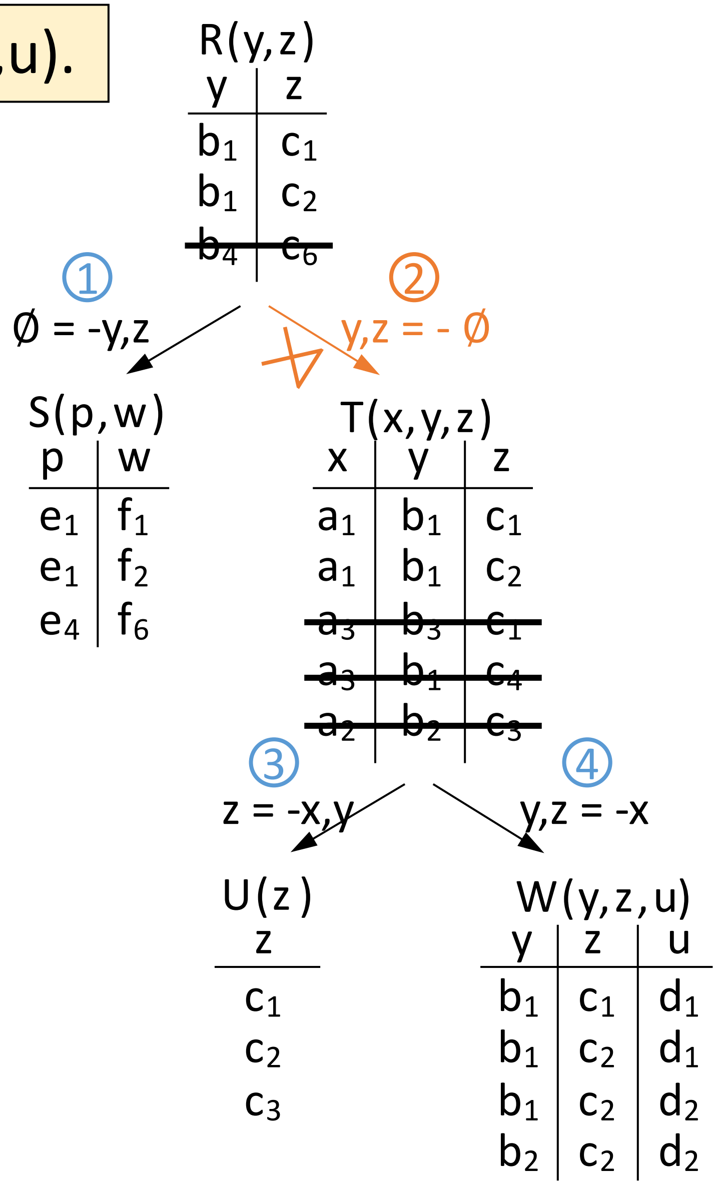
y	z	u
b_1	c_1	d_1
b_1	c_2	d_1
b_1	c_2	d_2
b_2	c_2	d_2

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**



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)

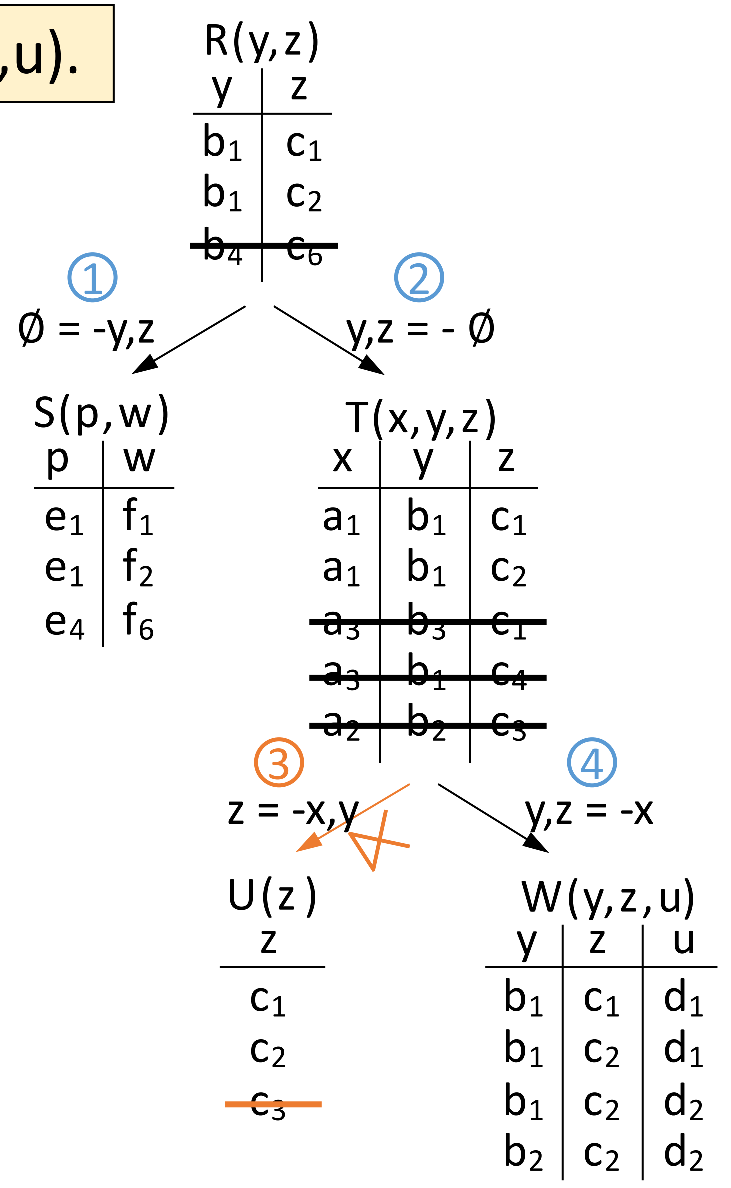
y	z	u
b_1	c_1	d_1
b_1	c_2	d_1
b_1	c_2	d_2
b_2	c_2	d_2

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**



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)

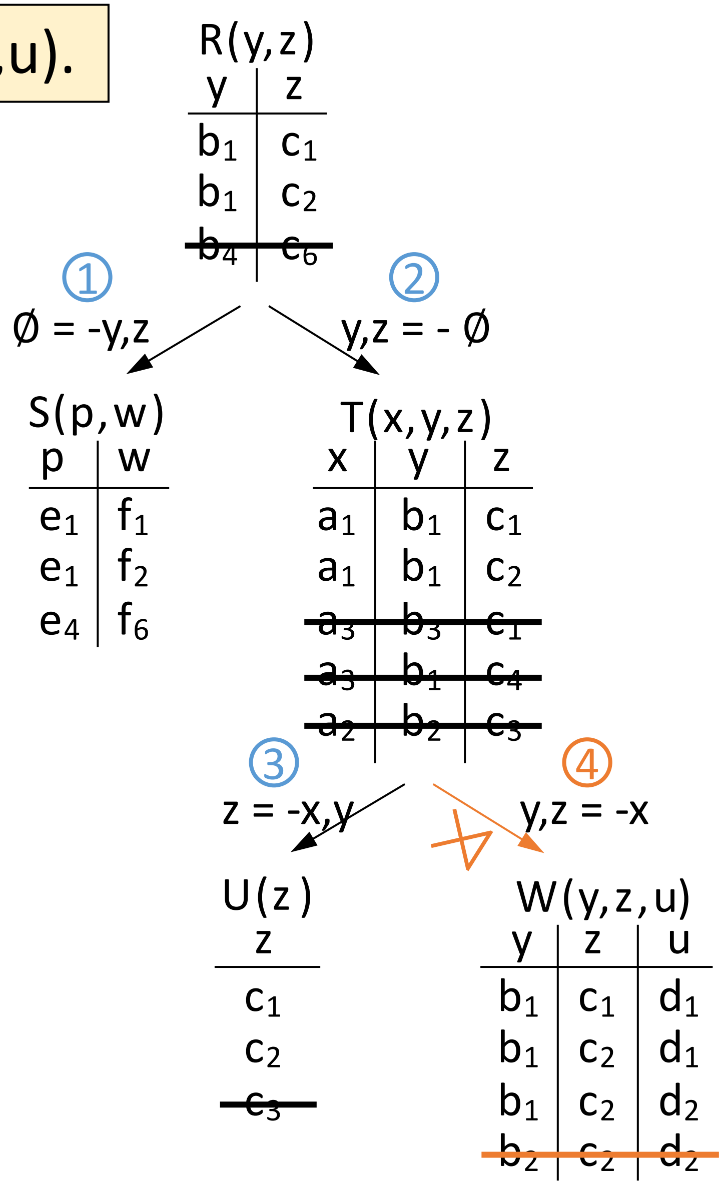
y	z	u
b_1	c_1	d_1
b_1	c_2	d_1
b_1	c_2	d_2
b_2	c_2	d_2

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 $\ltimes$ (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**



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

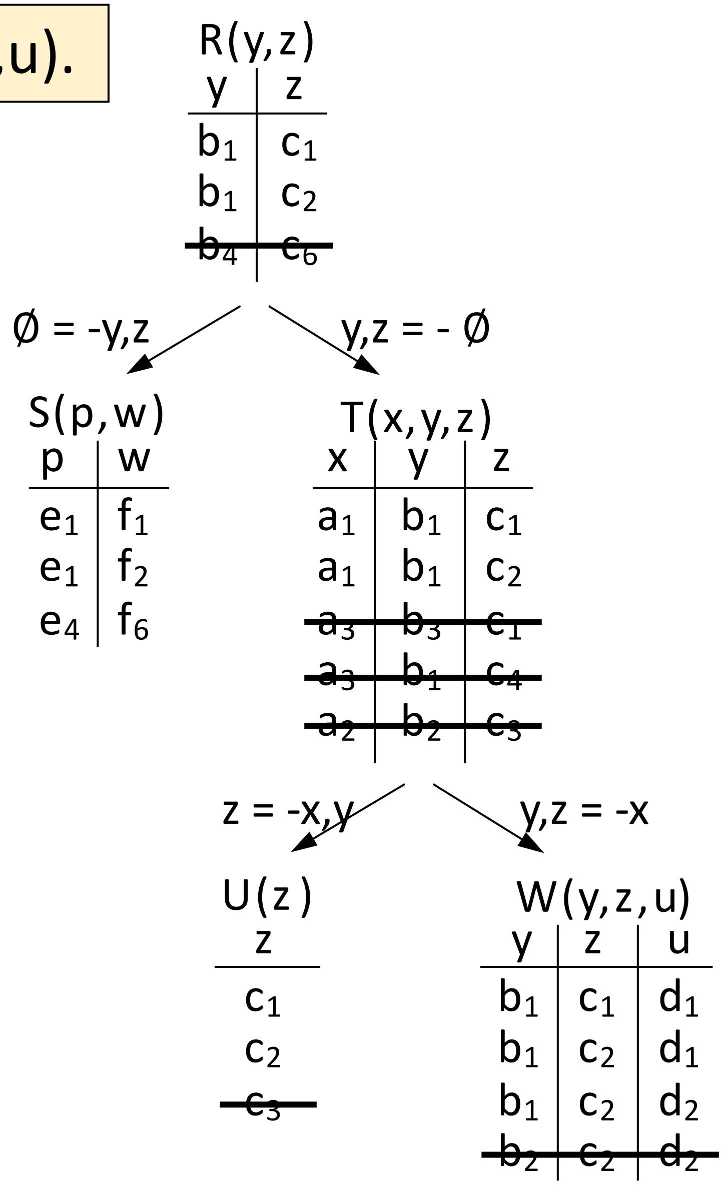
Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).

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**

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.



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: 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).

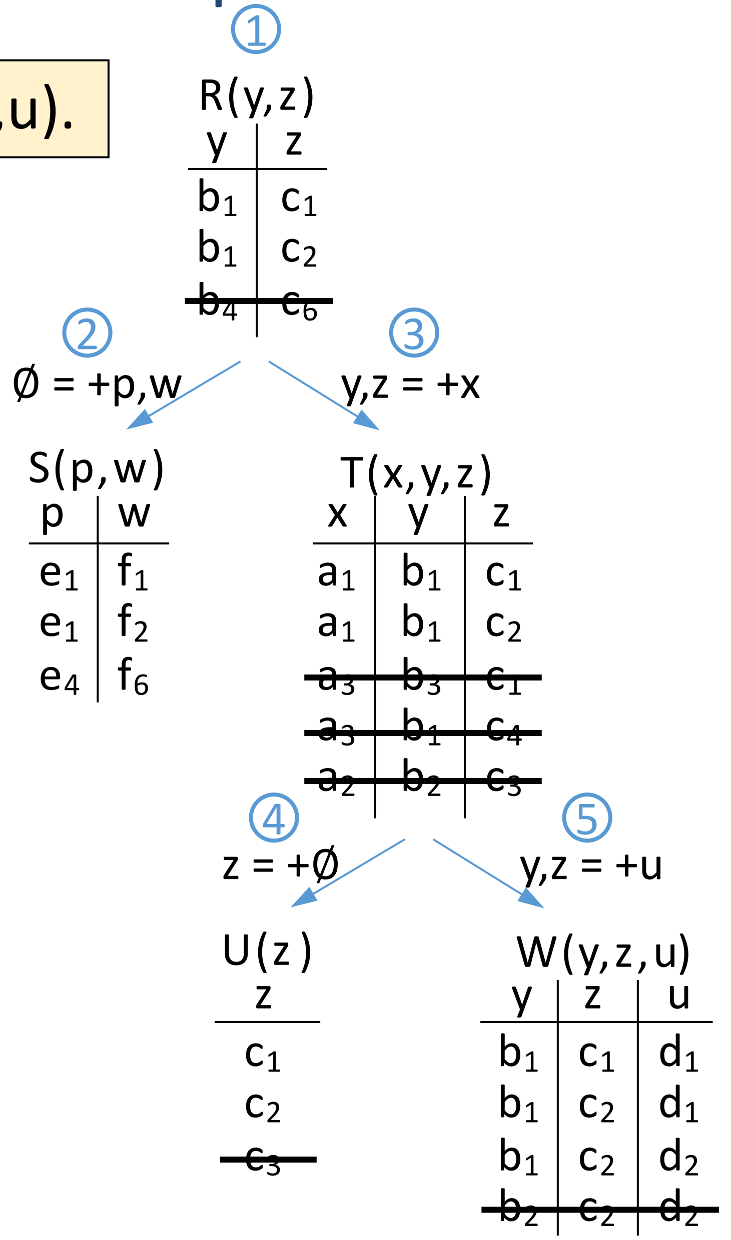
Semi-join phase ⋊ (remove dangling tuples) in *O*(IN)

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

Join phase ⋈ (compute results) in *O*(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
~~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~~

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).

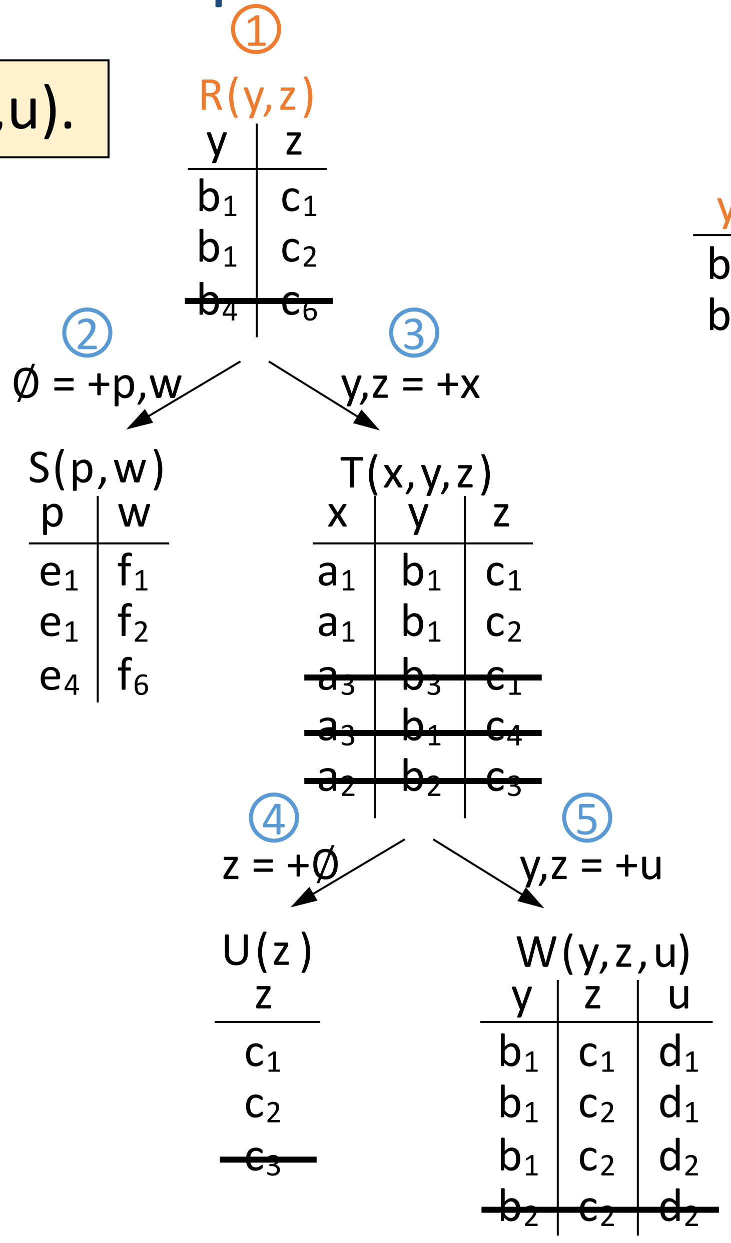
Semi-join phase ⋊ (remove dangling tuples) in *O*(IN)

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

Join phase ⋈ (compute results) in *O*(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
~~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~~

Join results

y	z
b_1	c_1
b_1	c_2

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).

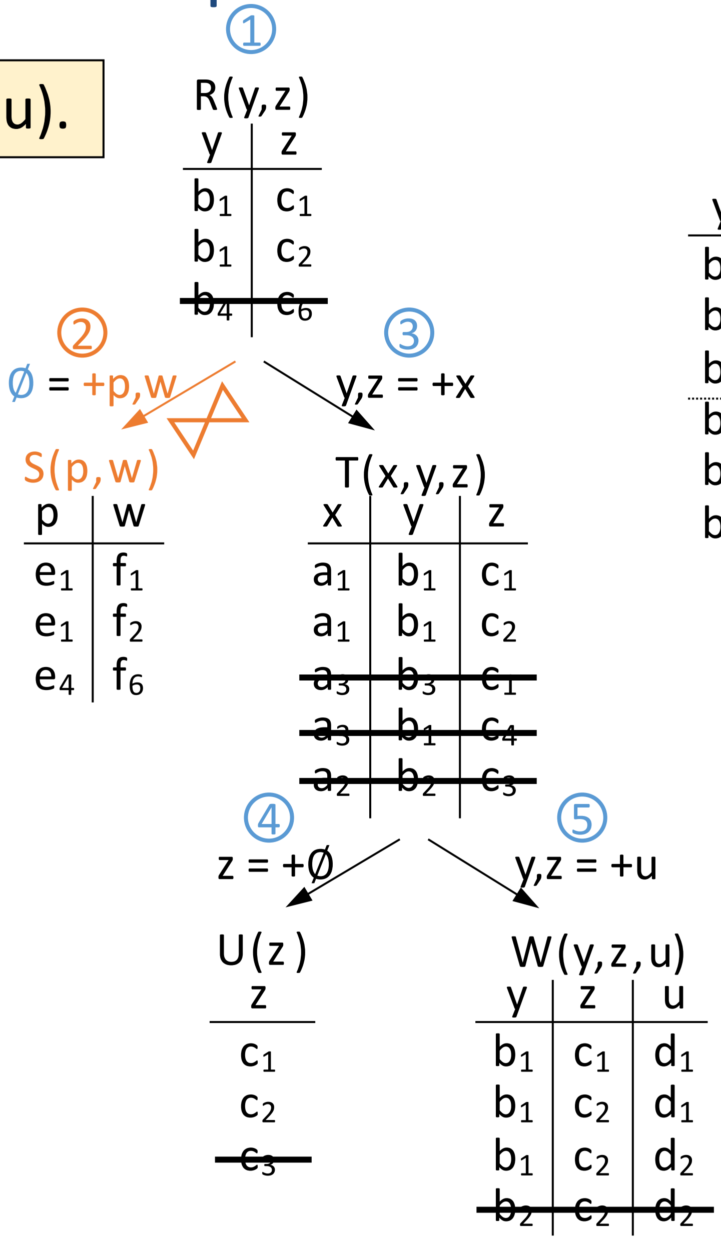
Semi-join phase ⋊ (remove dangling tuples) in *O*(IN)

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

Join phase ⋈ (compute results) in *O*(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
~~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~~

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).

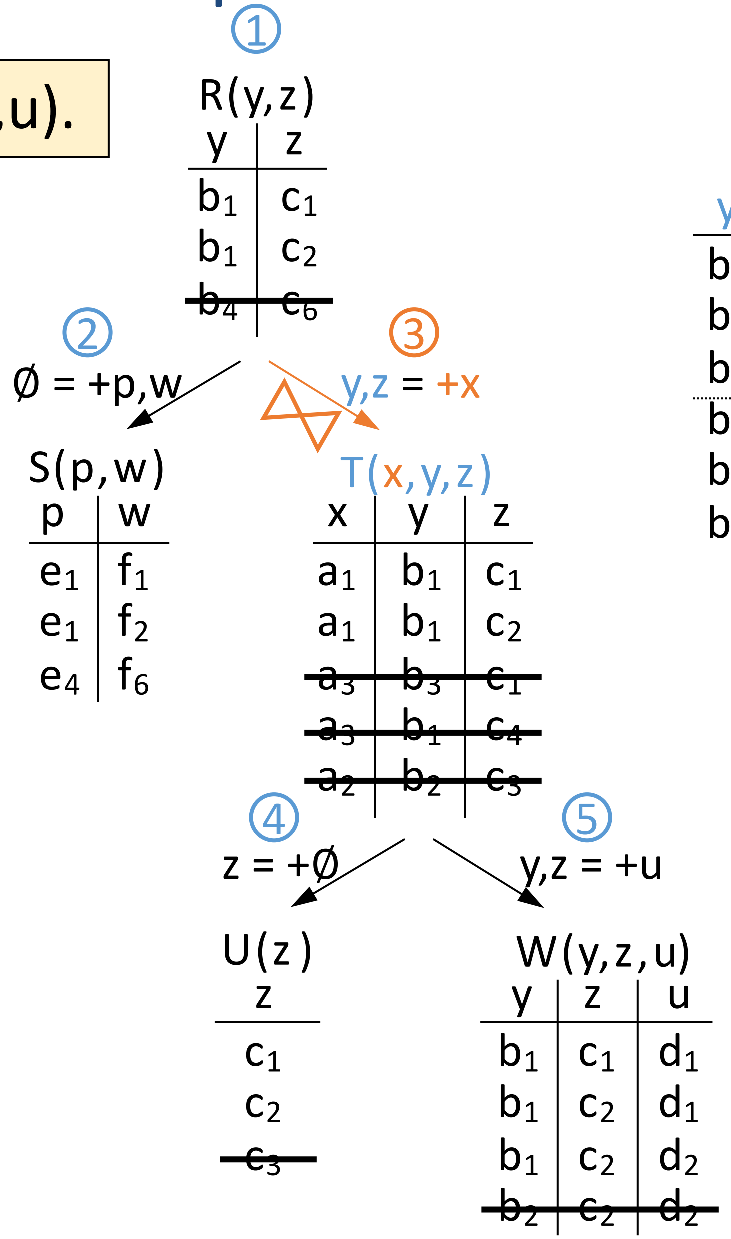
Semi-join phase ⋊ (remove dangling tuples) in *O*(IN)

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

Join phase ⋈ (compute results) in *O*(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
~~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~~

Join results

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).

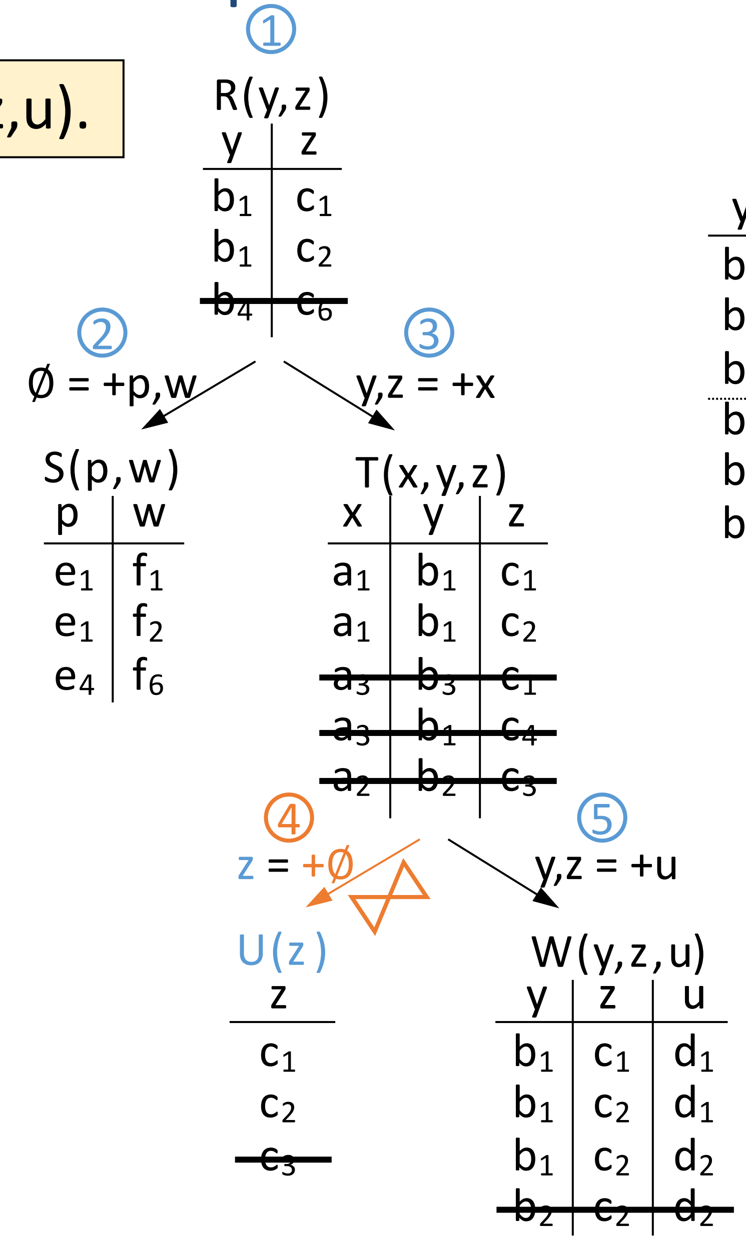
Semi-join phase ⋊ (remove dangling tuples) in O(IN)

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

Join phase ⋈ (compute results) in O(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
~~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~~

Join results

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).

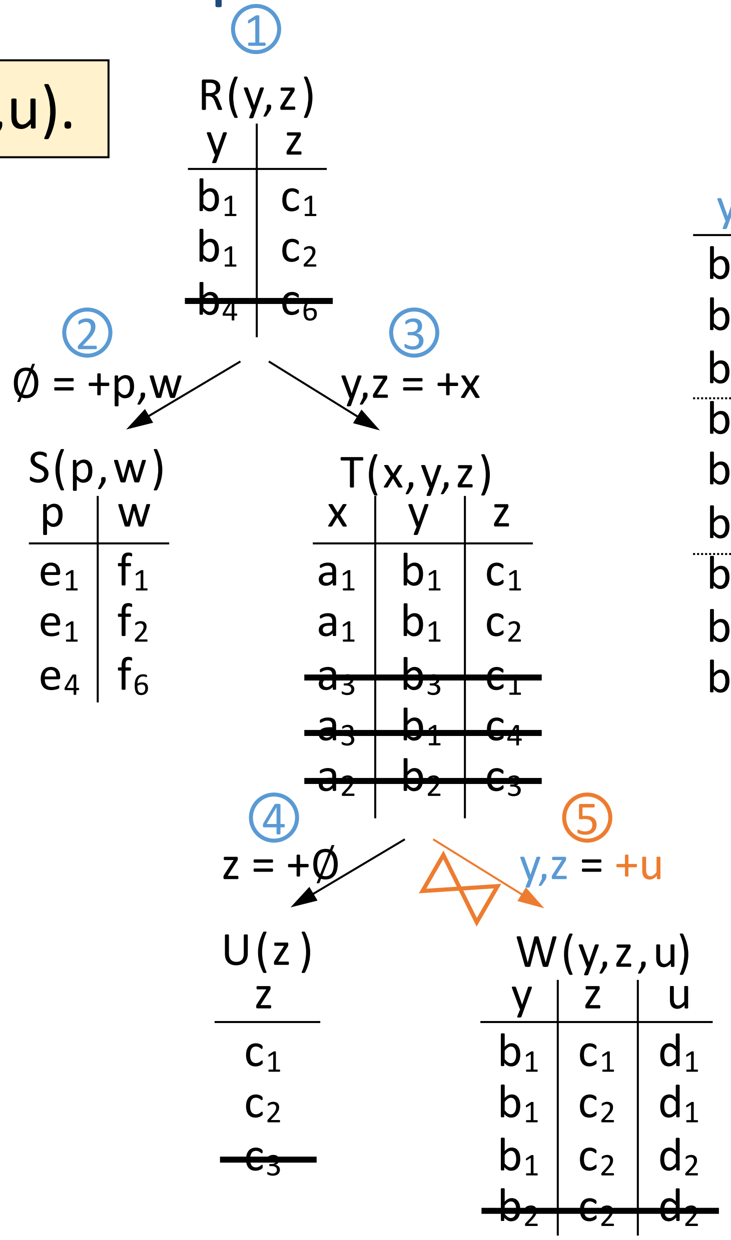
Semi-join phase ⋊ (remove dangling tuples) in *O*(IN)

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

Join phase ⋈ (compute results) in *O*(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
~~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~~

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

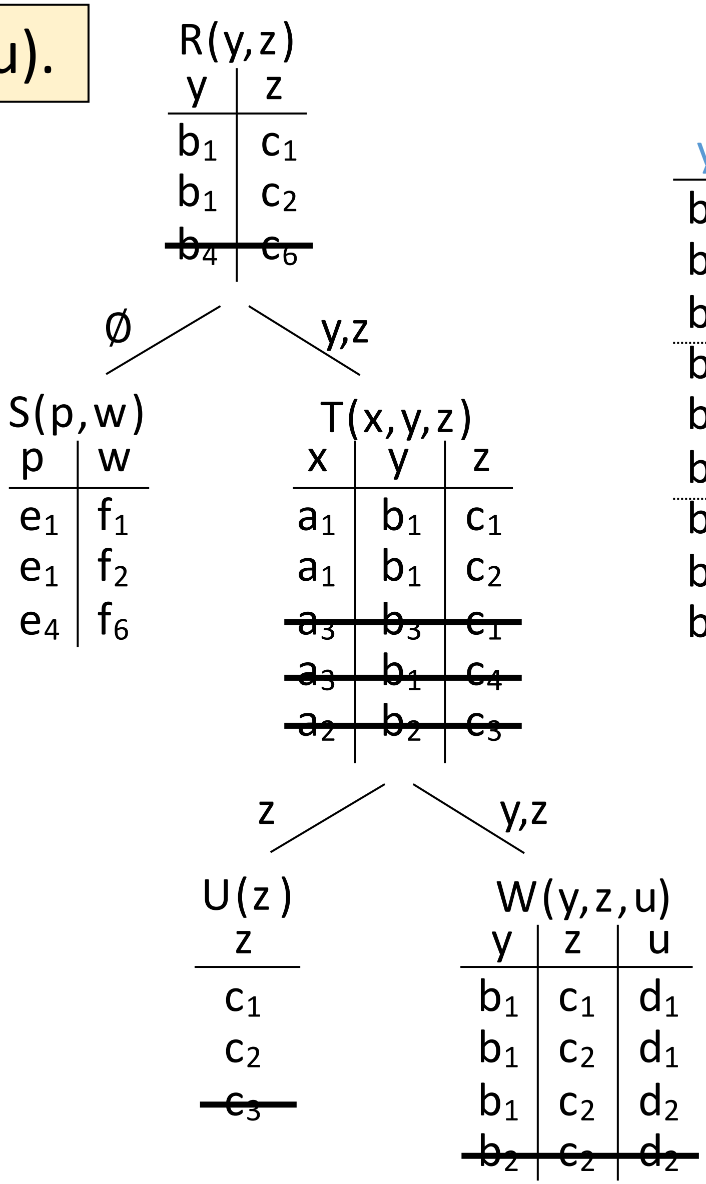
Q(y,z,p,w,x,u) :- R(y,z), S(p,w), T(x,y,z), U(z), W(y,z,u).

Semi-join phase ⋉ (remove dangling tuples) in *O*(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*(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
~~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~~

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) = Ø as follows:
  - Pick a root and assign to each R_j ∈ 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) = Ø, else Q(D) ≠ Ø.
- Theorem:
  - For ACQs Q: Deciding Q(D) = Ø 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