Updated 3/26/2022

Topic 3: Efficient query evaluation
Unit 1: Acyclic query evaluation
Lecture 17

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp22)

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

3/22/2022

Topic 1: Data Models and Query Languages

- **Lecture 1 (Tue 1/18):** Course introduction / T1-U1 SQL / PostgreSQL setup / SQL Activities
- **Lecture 2 (Fri 1/21):** T1-U1 SQL
- **Lecture 3 (Tue 1/25):** T1-U1 SQL
- **Lecture 4 (Fri 1/28):** T1-U1 SQL, T1-U2 Logic & Relational Calculus
- **Lecture 5 (Tue 2/1):** T1-U1 Logic & Relational Calculus
- **Lecture 6 (Fri 2/4):** T1-U3 Relational Algebra & Codd's Theorem
- **Lecture 7 (Tue 2/8):** T1-U3 Relational Algebra & Codd's Theorem
- **Lecture 8 (Fri 2/11):** T1-U3 Relational Algebra & Codd's Theorem / T1-U4 Datalog & Recursion
- **Lecture 9 (Tue 2/15):** T1-U4 Datalog & Recursion
- **Lecture 10 (Tue 2/18):** T1-U4 Datalog & Recursion
- **Lecture 11 (Tue 2/22):** T1-U4 Datalog & Recursion

Topic 2: Complexity of Query Evaluation & Reverse Data Management

- **CONTINUED Lecture 11 (Tue 2/22):** T2-U1 Conjunctive Queries
- **Lecture 12 (Fri 2/25):** T2-U1 Conjunctive Queries
- **Lecture 13 (Tue 3/1):** T2-U1 Conjunctive Queries / T2-U2 Beyond Conjunctive Queries
- **Lecture 14 (Fri 3/4):** T2-U3 Provenance
- **Lecture 15 (Tue 3/8):** T2-U3 Provenance
- **Lecture 16 (Fri 3/11):** T2-U3/U4 Provenance & Reverse Data Management

Topic 3: Efficient Query Evaluation & Factorized Representations

- **CONTINUED Lecture 16 (Fri 3/11):** T3-U1 Acyclic Queries
- Spring break
- **Lecture 17 (Tue 3/22):** Acyclic Queries
- **Lecture 18 (Fri 3/25):** Cyclic Queries
- **Lecture 19 (Tue 3/29):** Cyclic Queries
- **Lecture 20 (Fri 4/1):** Factorized Representations
- **Lecture 21 (Tue 4/5):** Factorized Representations
- **Lecture 22 (Fri 4/8):** Top-k & Optimization Problems
- **Lecture 23 (Tue 4/12):** Top-k & Optimization Problems

Pre-class conversations

- Recapitulation of Provenance and intro to trees (acyclic queries)
- Suggestion: Scribes with 2 iterations

- Today:
  - Acyclic queries, Yannakakis, Hypergraphs, GYO reduction
- Next time:
  - cycles, cycles, cycles

How Can Reasoners Simplify Database Querying (And Why Haven't They Done It Yet)?

Michael Benedikt
University of Oxford

March 30, 3pm, Datalab seminar:
https://db.khoury.northeastern.edu/activities/

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

Acyclic Conjunctive Queries

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

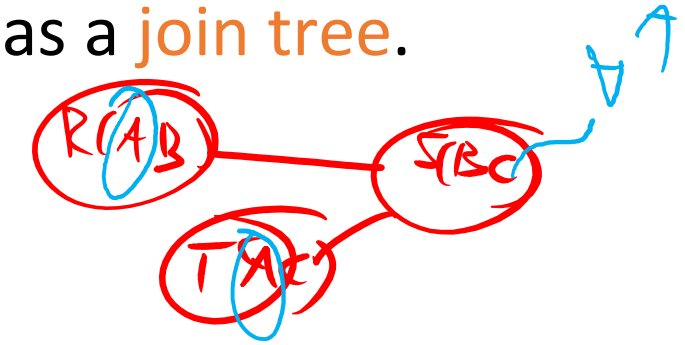
- Definition: A conjunctive query Q is (alpha) acyclic if it has a join tree.
- Let Q be a conjunctive query of the form

  $Q(\mathbf{x}) : \exists \mathbf{y}\ [R_1(\mathbf{z}_1) \wedge R_2(\mathbf{z}_2) \wedge \ldots \wedge R_m(\mathbf{z}_m)]$

  $Q(\mathbf{x}) :- R_1(\mathbf{z}_1), R_2(\mathbf{z}_2), \ldots, R_m(\mathbf{z}_m)$

$x \in Y \subseteq U z_i$

$Y \subseteq U z_i$

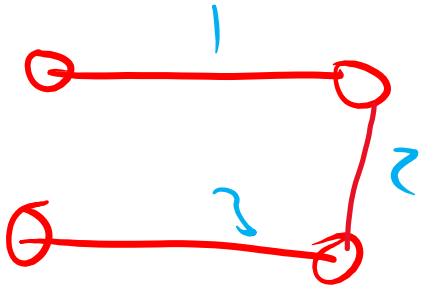


Notice that the definition of join tree does not include the root. We do need to choose roots later, and different choices of roots lead to different sequences of semijoin reductions

- A join tree for Q is a tree T =(V,E) such that:
  - V: The nodes of T are the atoms $R_i(\mathbf{z}_i)$, $1 \le i \le m$, of Q.
  - E: For every variable w occurring in Q, the set of the nodes of T that contain w forms a (connected) subtree of T;
    - $\{R_j(\mathbf{z}_j) \in V \mid z \text{ occurs in } R_j(\mathbf{z}_j)\}$ induces a connected subtree in T
    - in other words, if a variable w occurs in two different atoms $R_j(\mathbf{z}_j)$ and $R_k(\mathbf{z}_k)$ of Q, then it occurs in each atom on the unique path of T joining $R_j(\mathbf{z}_j)$ and $R_k(\mathbf{z}_k)$
    - also called: running intersection property (see also junction tree algorithm and tree decompositions)

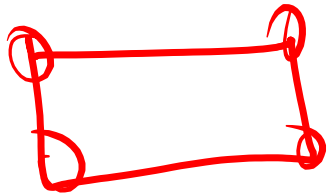
Which query is acyclic?

- Path of length 3? (4 nodes, 3 edges). Return end points.



?

- Cycle of length 4 ? Boolean.

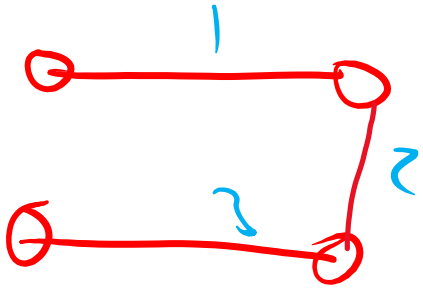


?

Which query is acyclic?

- Path of length 3? (4 nodes, 3 edges). Return end points.

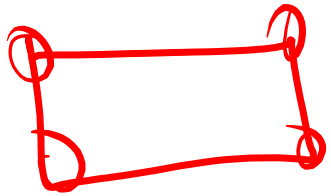
$P4(x_1,x_4)$:- $E(x_1,x_2)$, $E(x_2,x_3)$, $E(x_3,x_4)$



?

- Cycle of length 4? Boolean.

$C4()$:- $E(x_1,x_2)$, $E(x_2,x_3)$, $E(x_3,x_4)$, $E(x_4,x_1)$

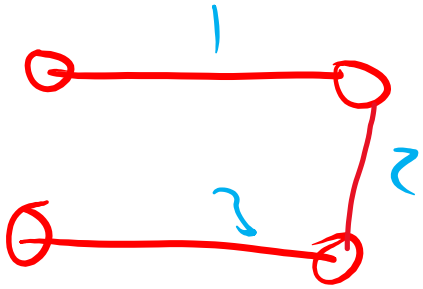


?

Which query is acyclic?

- Path of length 3? (4 nodes, 3 edges). Return end points.

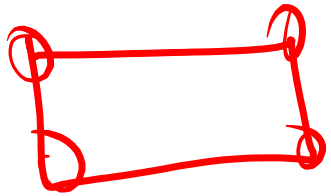
$P4(x_1,x_4)$:- $E(x_1,x_2)$, $E(x_2,x_3)$, $E(x_3,x_4)$



acyclic
(join tree is path)

- Cycle of length 4? Boolean.

$C4()$:- $E(x_1,x_2)$, $E(x_2,x_3)$, $E(x_3,x_4)$, $E(x_4,x_1)$

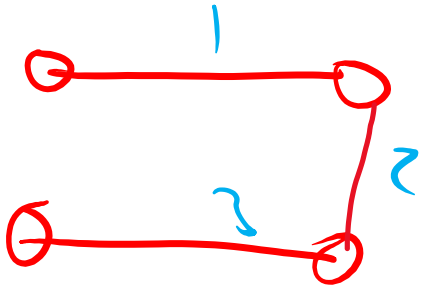


?

Which query is acyclic?

- Path of length 3? (4 nodes, 3 edges). Return end points.

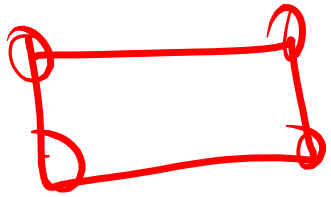
$P4(x_1,x_4) :- E(x_1,x_2), E(x_2,x_3), E(x_3,x_4)$



acyclic
(join tree is path)

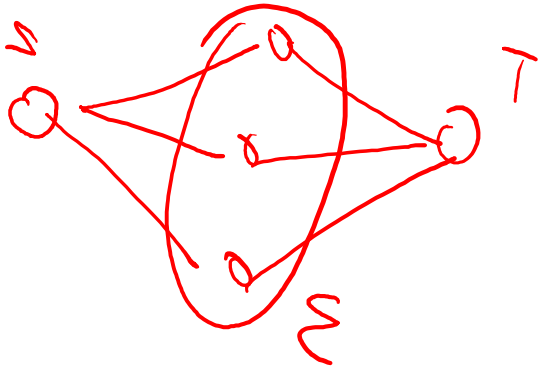
- Cycle of length 4? Boolean.

$C4() :- E(x_1,x_2), E(x_2,x_3), E(x_3,x_4), E(x_4,x_1)$



cyclic

Acyclic Conjunctive Queries



- Is the following query acyclic?

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

Dual Hypergraph (relations as nodes)

?

Hypergraph (variables as nodes)

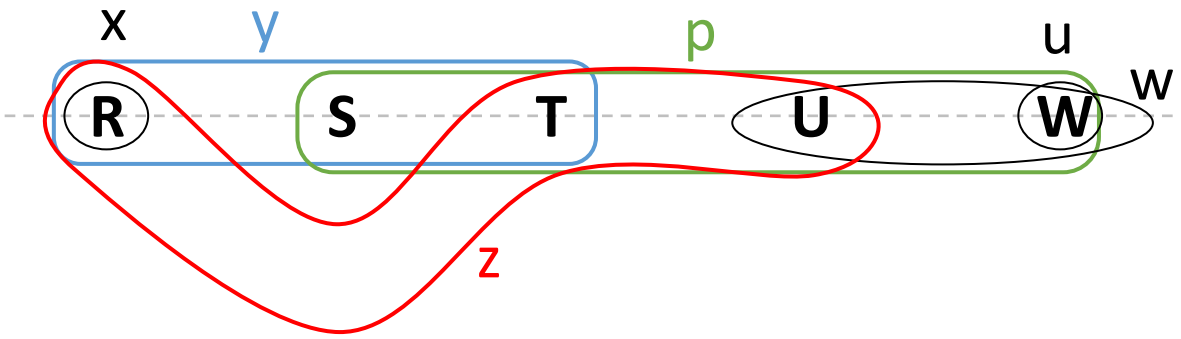
?

Acyclic Conjunctive Queries

- Is the following query acyclic?

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

Dual Hypergraph (relations as nodes)



Hypergraph (variables as nodes)

?

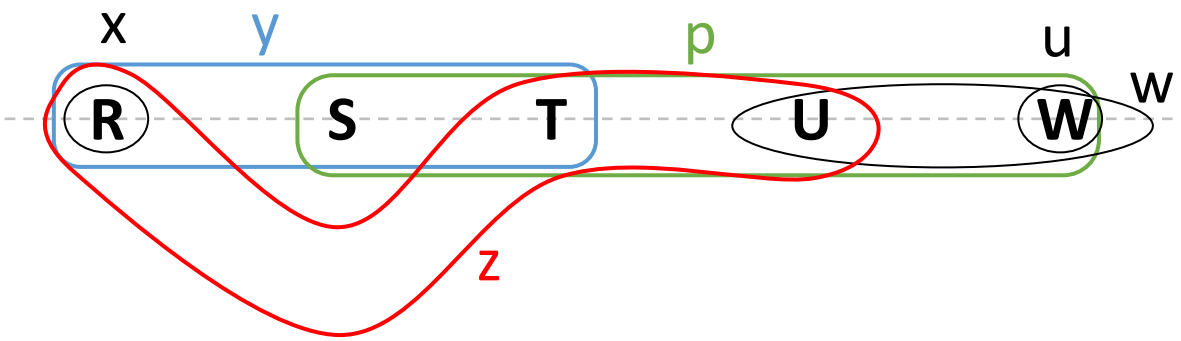
No linear order in dual hypergraph ☹
(Thus resilience is NPC!)

Acyclic Conjunctive Queries

- Is the following query acyclic?

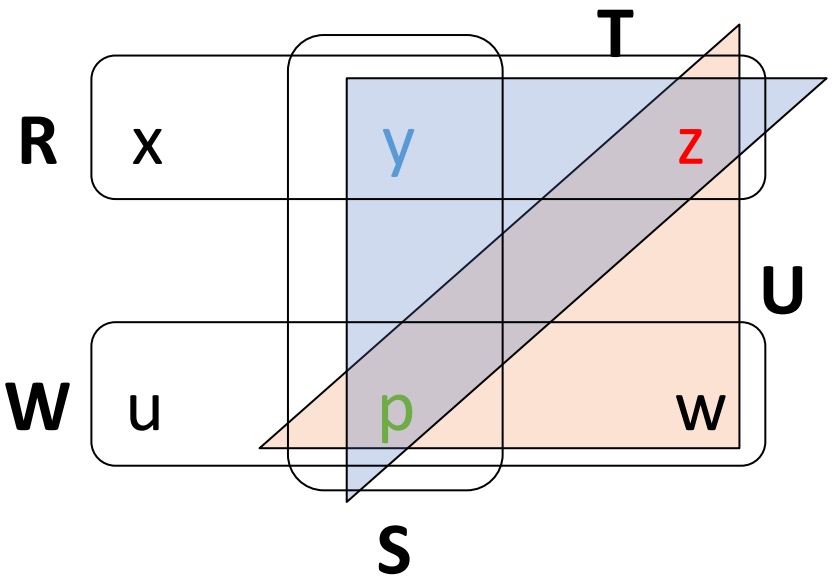
Q :- R(x,y,z), S(y,p), T(y,z,p), U(z,p,w), W(p,w,u).

Dual Hypergraph (relations as nodes)



No linear order in dual hypergraph ☹
(Thus resilience is NPC!)

Hypergraph (variables as nodes)

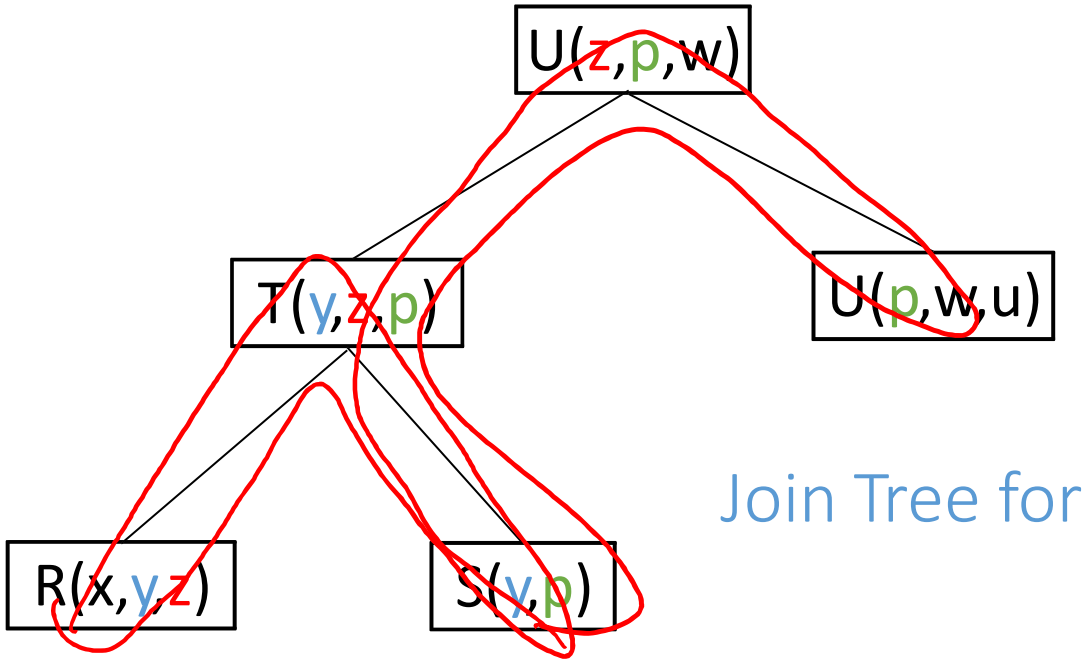


So does the query have a join tree?

Acyclic Conjunctive Queries

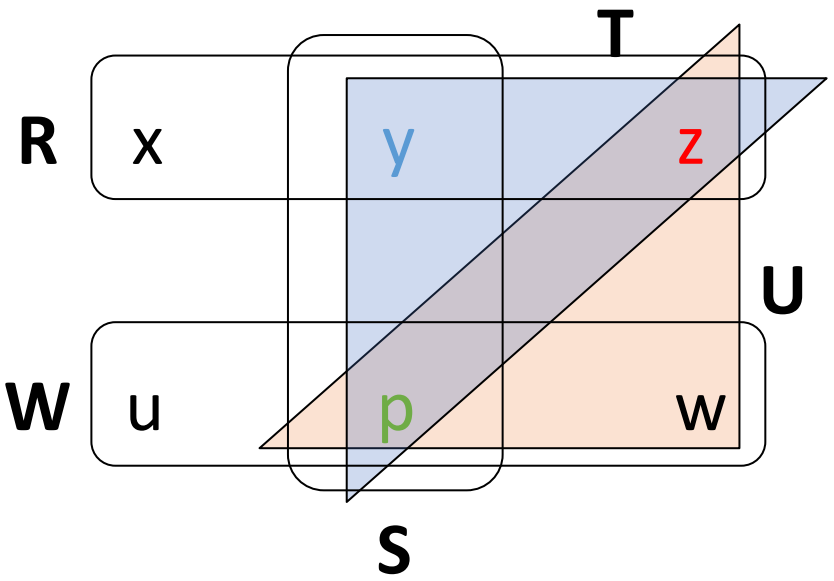
- Is the following query acyclic?

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



Join Tree for Q

Hypergraph (variables as nodes)



Yes the query has a join tree

[Yannakakis'81] Acyclic Conjunctive Queries

- Theorem [Yannakakis'81]:
  - The Acyclic Conjunctive Query Evaluation Problem is tractable.
  - There is an algorithm for query evaluation with following properties:
    - If Q is a Boolean acyclic conjunctive query,
      then the algorithm runs in time: $O(|Q| \cdot |D|)$
    - If Q is a full (i.e. no projections) acyclic conjunctive query
      then the algorithm runs in time: $O(|Q| \cdot |D| + |Q(D)|)$
  - In terms of data complexity, this means $O(|Input| + |Output|)$
    - i.e., it runs in input/output linear time, which is the "right" notion of tractability in this case (you can't do better, in general)

Yannakakis, "Algorithms for acyclic database schemes", VLDB 1981. https://dl.acm.org/doi/10.5555/1286831.1286840
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Yannakakis' Algorithm for Boolean acyclic CQs

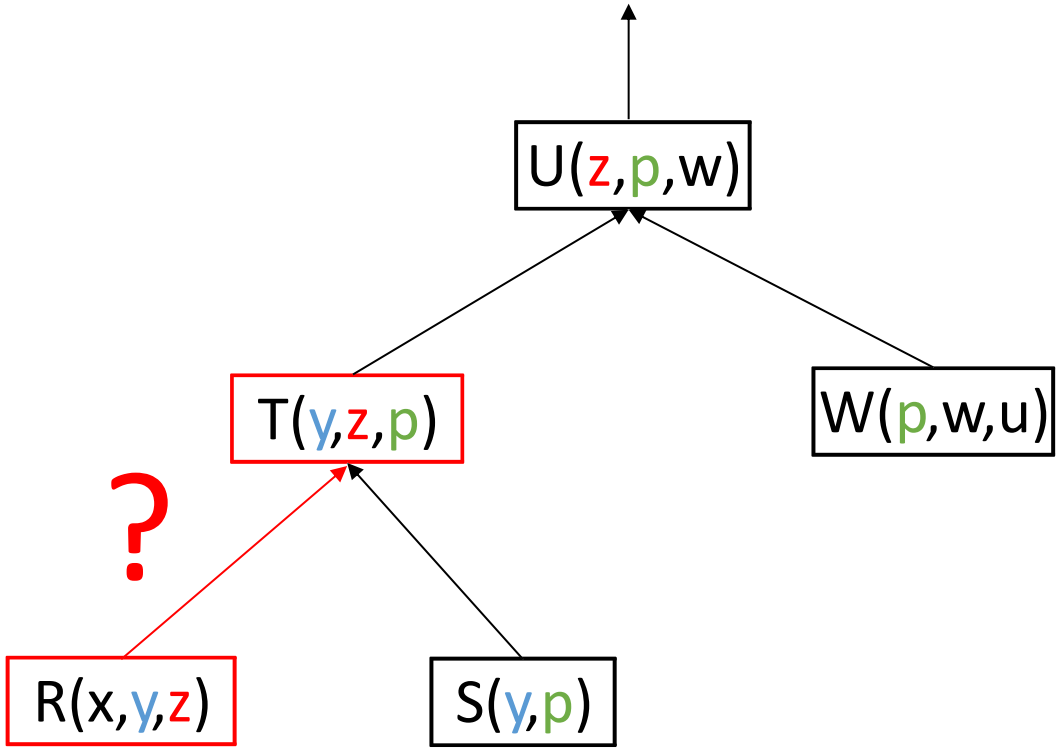
Basically the standard Dynamic Programming Algorithm

- Input: Boolean acyclic conjunctive query Q, database D
- Construct a join tree T of Q
- Populate the nodes of T with the matching relations of D.
- Traverse the tree T bottom up (also called "collection phase"):
  - For each node compute the semi-joins of the (current) relation in the node with the (current) relations in the children of the node
- Examine the resulting relation R at the root of T
  - If R is non-empty, then output Q(D) = 1 (D satisfies Q).
  - If R is empty, then output Q(D) = 0 (D does not satisfy Q).

Acyclic Conjunctive Queries

- Where are the semi-join reductions in following query:

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



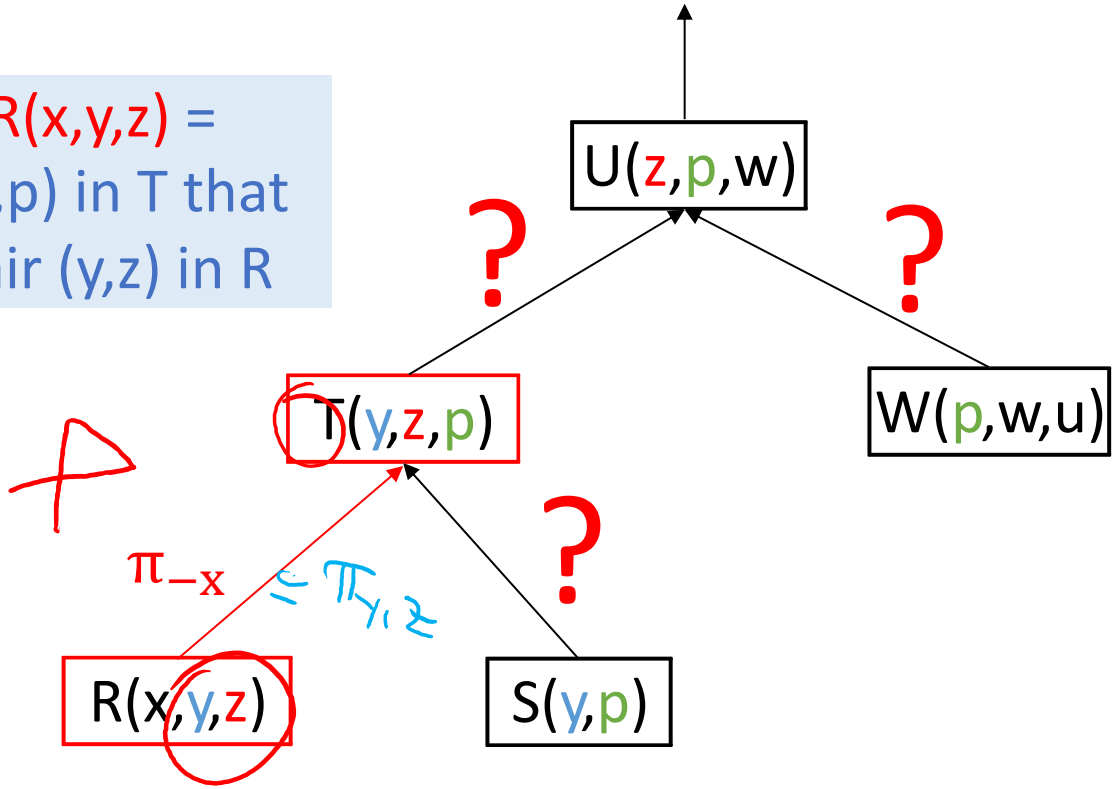
Rooted Join Tree for Q

Acyclic Conjunctive Queries

- Where are the semi-join reductions in following query:

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

T(y,z,p) ⋉ R(x,y,z) =
all triples (y,z,p) in T that
"match" a pair (y,z) in R



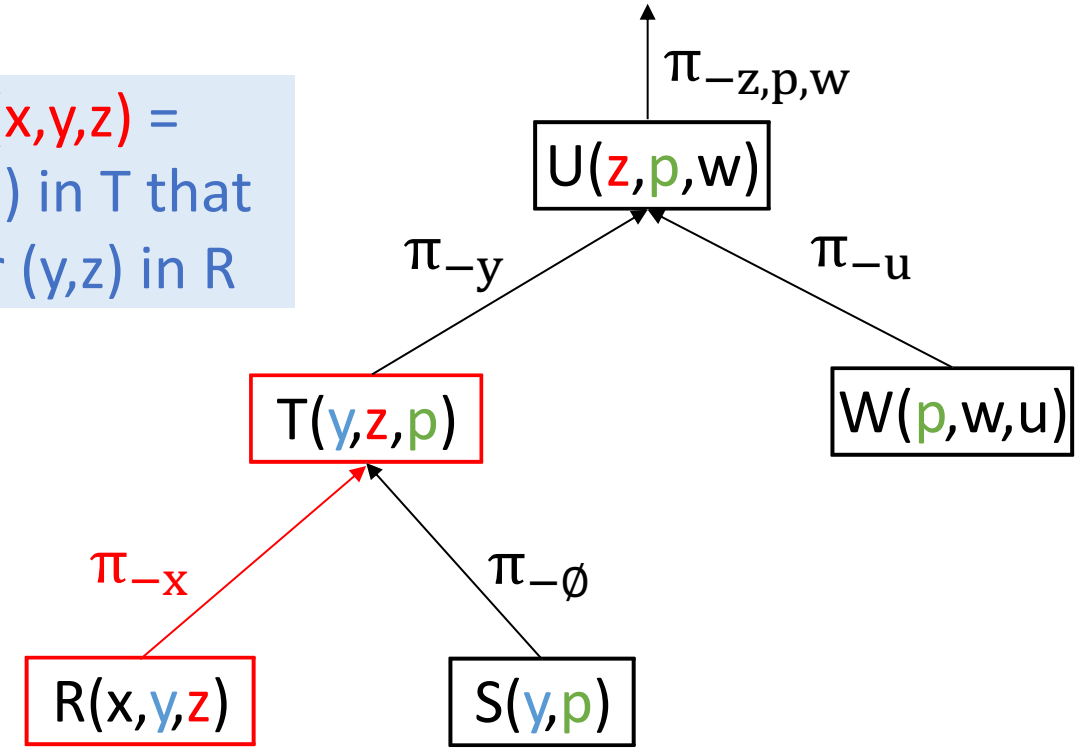
Rooted Join Tree for Q

Acyclic Conjunctive Queries

- Where are the semi-join reductions in following query:

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

T(y,z,p) ⋉ R(x,y,z) =
all triples (y,z,p) in T that
"match" a pair (y,z) in R



Rooted Join Tree for Q

More on Yannakakis' Algorithm

- The join tree makes it possible to avoid exponential explosion (in size |Q|) of the size of the intermediate computations.
- The algorithm can be extended to non-Boolean conjunctive queries using one additional top-down traversal of the join tree.
  - Bottom up (or "collect")
  - Top down (or "distribute")
- There are efficient algorithms for detecting acyclicity & computing a join tree.
  - [Tarjan, Yannakakis'84] Linear-time algorithm for detecting acyclicity and computing a join tree.
  - [Gottlob, Leone, Scarcello'01 (FOCS'98)] Detecting acyclicity is LOGCFL-complete and thus highly parallelizable)

Tarjan, Yannakakis. "Simple Linear-Time Algorithms to Test Chordality of Graphs, Test Acyclicity of Hypergraphs, and Selectively Reduce Acyclic Hypergraphs", SIAM J. Comput. 1984. https://doi.org/10.1137/0213035 / Gottlob, Leone, Scarcello. "The Complexity of Acyclic Conjunctive Queries", JACM 2001. https://doi.org/10.1145/382780.382783

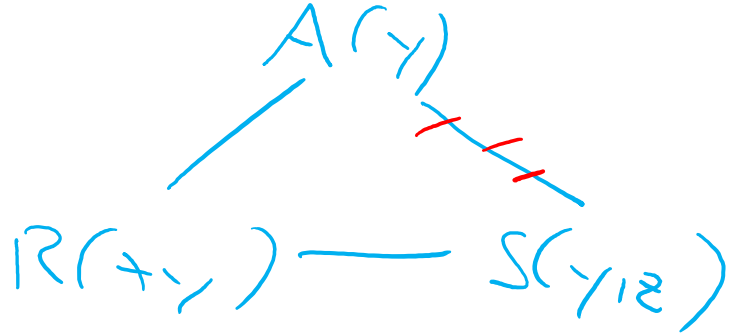
Are join trees unique?

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

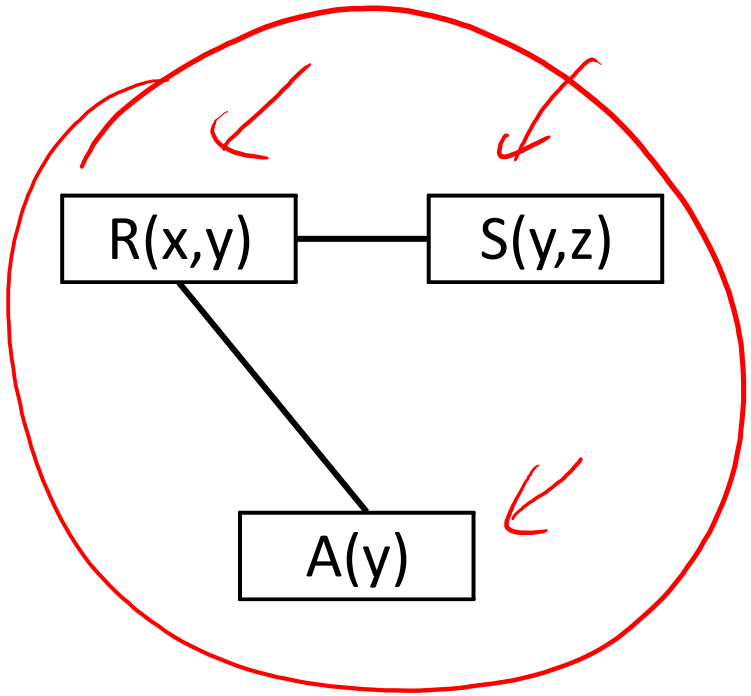
?

Are join trees unique?

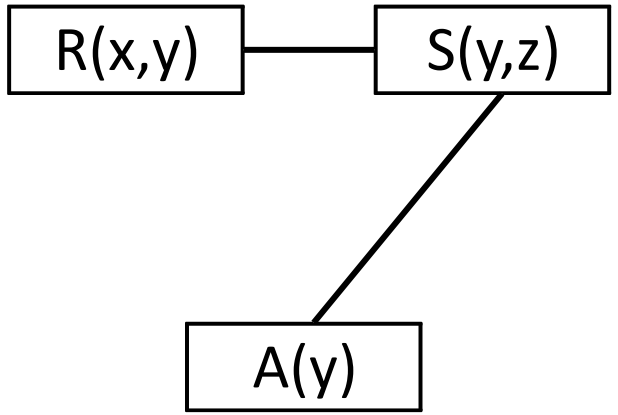
Q :- R(x,y), S(y,z), A(y).

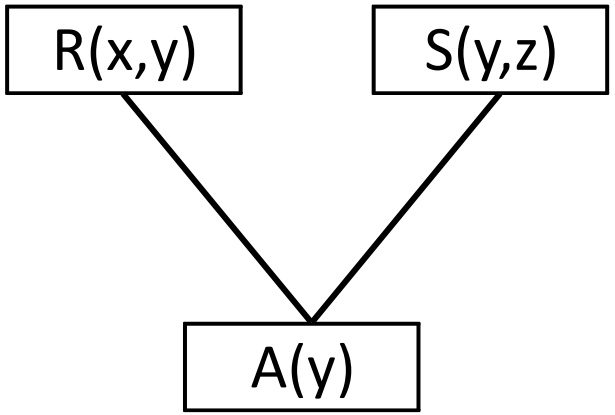


3 CHOICES FOR ROOT



In addition, we have 3 choices as roots for each of the 3 different join trees.
Leads to 9 different rooted join trees!





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

Query Hypergraph (vs. Dual Hypergraph vs. Incidence Matrix)

$$q :- A(x)S_1(x,v)S_2(v,y)B(y,u)S_3(y,z)D(z,w)C(z)$$

Query hypergraph

?

Incidence matrix

?

Query dual hypergraph

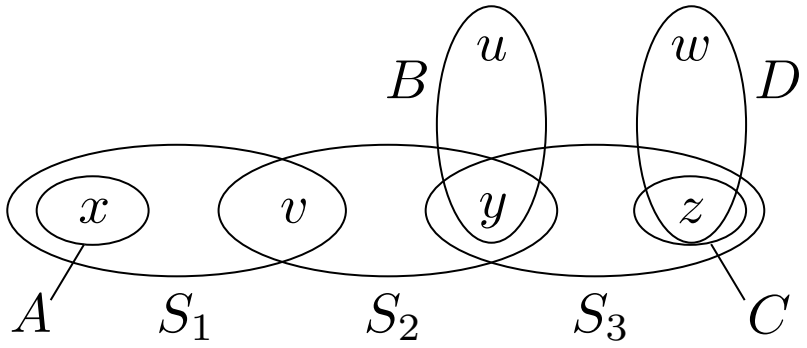
?

Query Hypergraph (vs. Dual Hypergraph vs. Incidence Matrix)

$$q :- A(x)S_1(x,v)S_2(v,y)B(y,u)S_3(y,z)D(z,w)C(z)$$

Query hypergraph

determines (alpha) acyclicity of CQs (next in class)



Incidence matrix

?

Query dual hypergraph

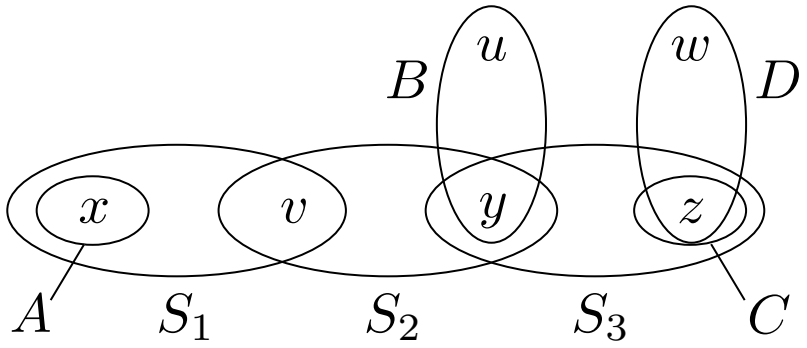
?

Query Hypergraph (vs. Dual Hypergraph vs. Incidence Matrix)

$$q :- A(x)S_1(x,v)S_2(v,y)B(y,u)S_3(y,z)D(z,w)C(z)$$

Query hypergraph

determines (alpha) acyclicity of CQs (next in class)

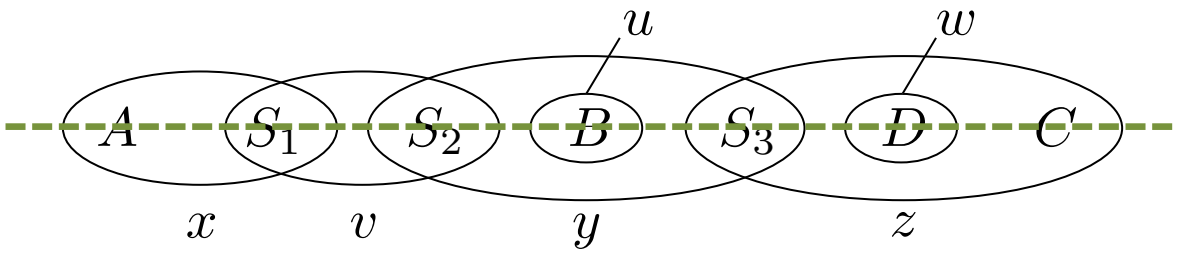


Incidence matrix

?

Query dual hypergraph

determines complexity of resilience [Meliou+'10]



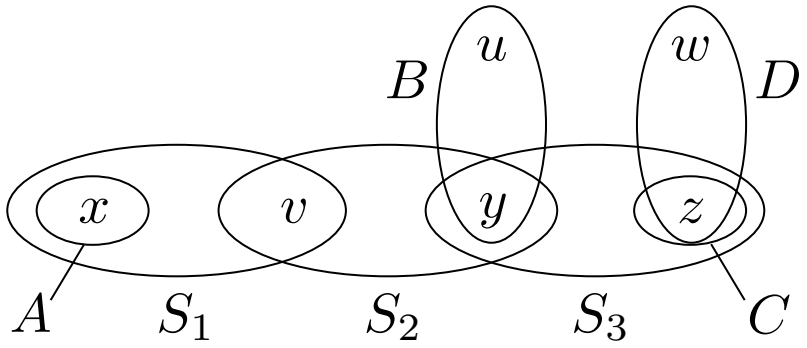
Meliou, Gatterbauer, Moore, Suciu. "The complexity of causality and responsibility for query answers and non-answers", PVLDB 2010. https://doi.org/10.14778/1880172.1880176 /
Gatterbauer, Suciu. "Dissociation and propagation for approximate lifted inference with standard relational database management systems", VLDBJ 2017. https://doi.org/10.1007/s00778-016-0434-5
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Query Hypergraph (vs. Dual Hypergraph vs. Incidence Matrix)

$$q :- A(x)S_1(x,v)S_2(v,y)B(y,u)S_3(y,z)D(z,w)C(z)$$

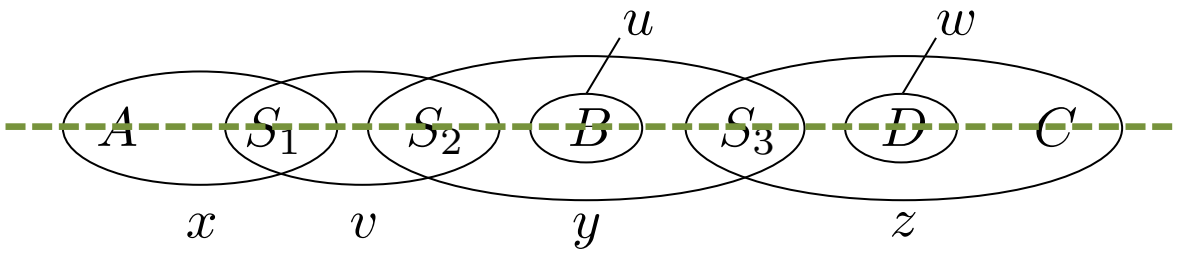
Query hypergraph

determines (alpha) acyclicity of CQs (next in class)



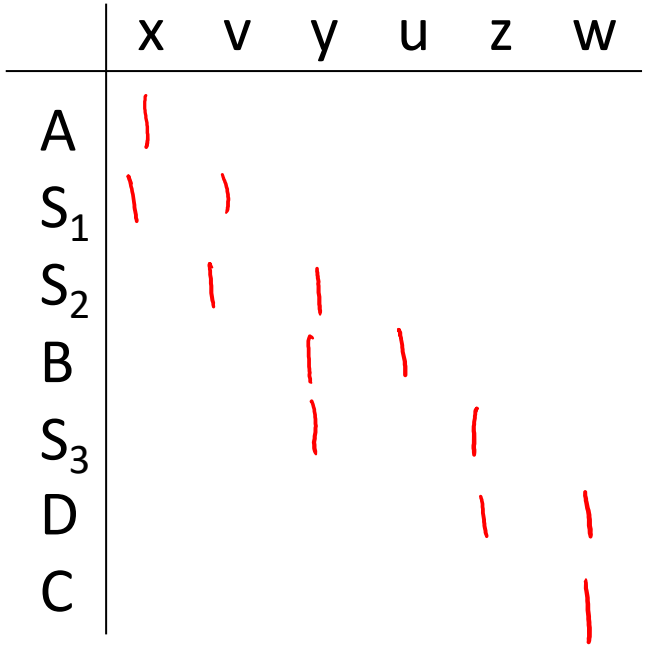
Query dual hypergraph

determines complexity of resilience [Meliou+'10]



Incidence matrix

determines minimal query plans [G+'17]
(intuitively, plans with early projections)



Meliou, Gatterbauer, Moore, Suciu. "The complexity of causality and responsibility for query answers and non-answers", PVLDB 2010. https://doi.org/10.14778/1880172.1880176 /
Gatterbauer, Suciu. "Dissociation and propagation for approximate lifted inference with standard relational database management systems", VLDBJ 2017. https://doi.org/10.1007/s00778-016-0434-5
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

GYO reduction (Graham-Yu-Ozsoyoglu) on the hypergraph

corresponds to a projection

- An ear is a hyperedge H whose variables (=nodes) form two groups:
  1. isolated variables that appear exclusively in H, and
  2. join variables (i.e. they occur in at least one other edge). For those join variables there exists a hyperedge called a witness that contains them all

corresponds to a join

- GYO algorithm (Ear removal)
  - Remove ears greedily from the hypergraph
- Important: any sequence of reductions that removes all hyperedges implies a join tree:
  - Just draw an edge between an ear and any witness (notice that if an ear has only isolated nodes, any remaining hyperedge is a witness)

This algorithm is named in honor of Marc H. Graham and the team Clement Yu and Meral Ozsoyoglu, who independently came to essentially this algorithm in [YO79] and [Gra79]:
[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. https://doi.org/10.1109/CMPSAC.1979.762509

Proof GYO reduction = acyclic query

- Proof (GYO ⇒ acyclic): if GYO leads to an empty hypergraph, then the resulting graph forms a valid join tree
  - notice that by construction, for any variable, those edges containing the variable will form an induced subtree

- Proof (acyclic ⇒ GYO): if there is a valid join tree, then removing leaf nodes in any order corresponds to a sequence of GYO reductions:
  - All non-exclusive variables from a leaf must be shared with the parent. Thus the parent forms a witness that consumes the leaf (notice that by construction, this also works if the leaf node shares no variables)

GYO reduction: Example 1

- GYO Ear removal

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

  - remove ears (= edges) greedily from hypergraph
    - 1. remove isolated nodes (also called singleton variables)
    - 2. remove consumed or empty edges (atoms)

Join tree

?

Query hypergraph

?

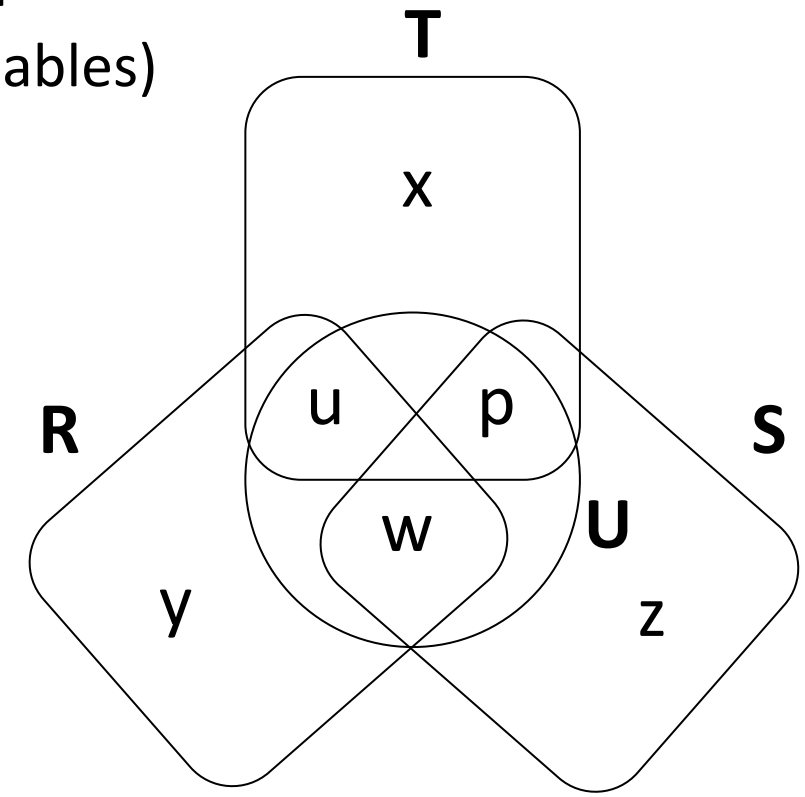
GYO reduction: Example 1

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

- GYO Ear removal
  - remove ears (= edges) greedily from hypergraph
    - 1. remove isolated nodes (also called singleton variables)
    - 2. remove consumed or empty edges (atoms)

Join tree

?

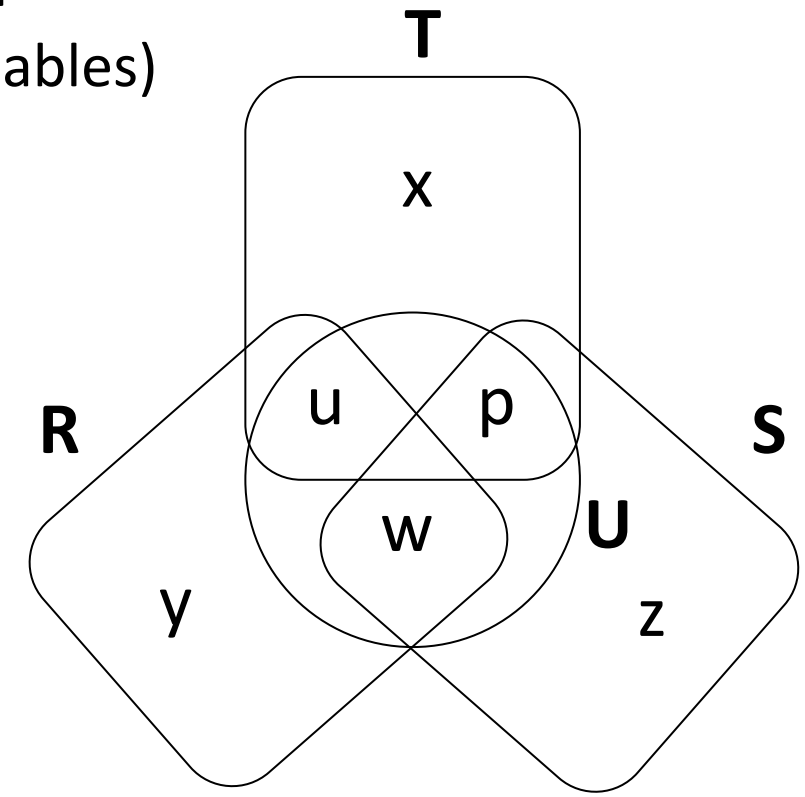


GYO reduction: Example 1

- GYO Ear removal

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

  - remove ears (= edges) greedily from hypergraph
    - 1. remove isolated nodes (also called singleton variables)
    - 2. remove consumed or empty edges (atoms)



Join tree ?

U(u,p,w)

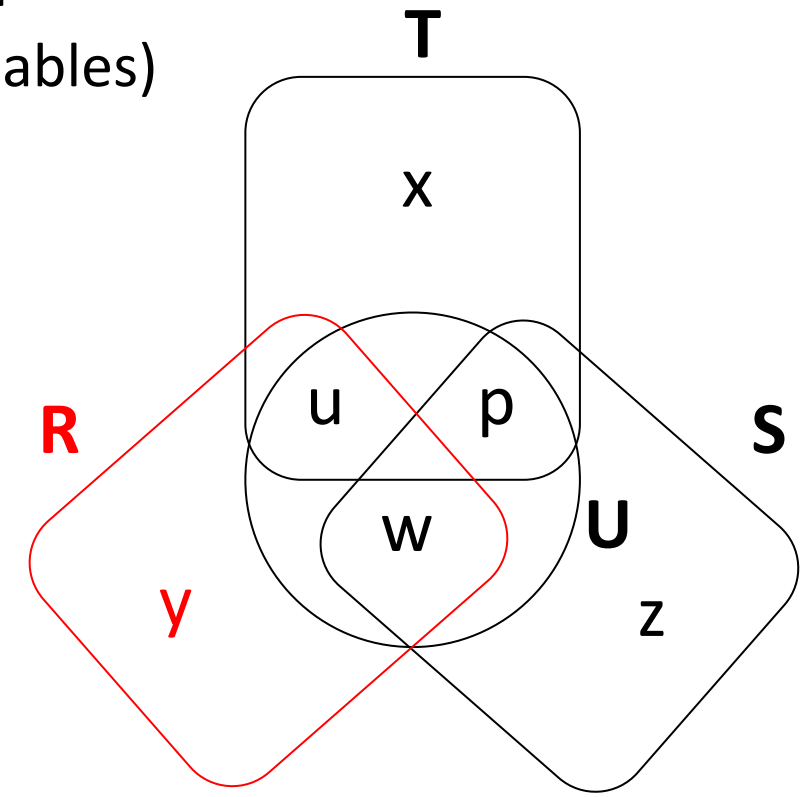
R(y,u,w) S(z,p,w) T(x,u,p)

GYO reduction: Example 1

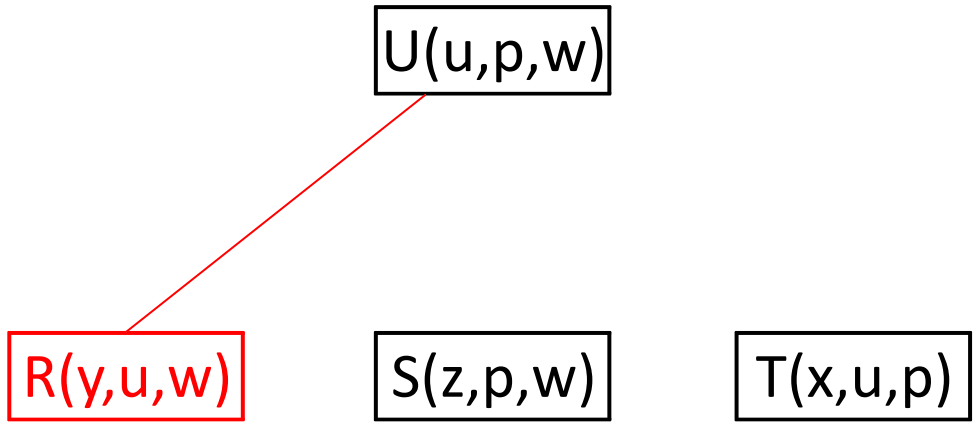
- GYO Ear removal

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

  - remove ears (= edges) greedily from hypergraph
    - 1. remove isolated nodes (also called singleton variables)
    - 2. remove consumed or empty edges (atoms)



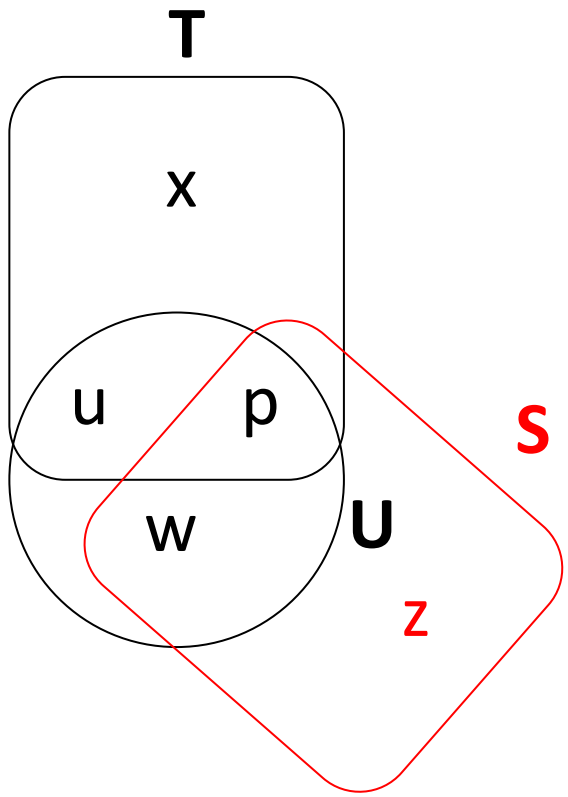
Join tree



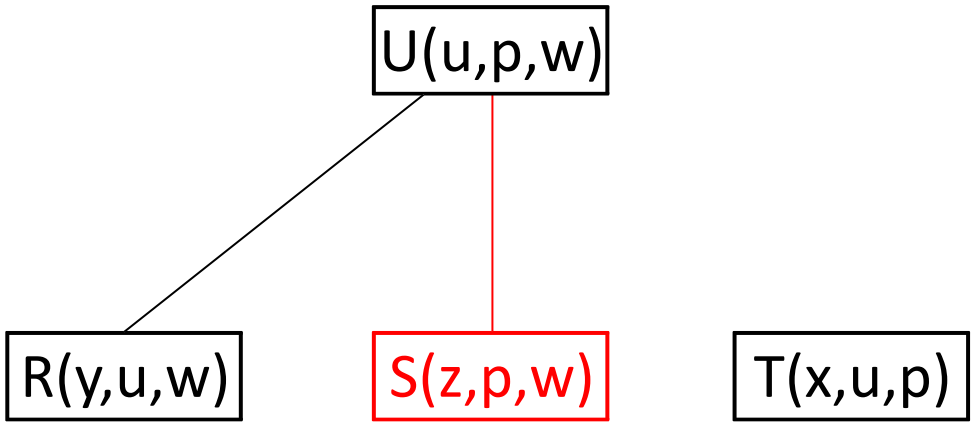
GYO reduction: Example 1

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

- GYO Ear removal
  - remove ears (= edges) greedily from hypergraph
    - 1. remove isolated nodes (also called singleton variables)
    - 2. remove consumed or empty edges (atoms)



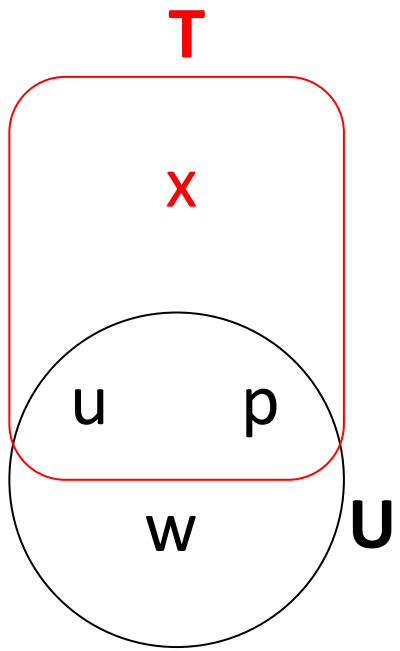
Join tree



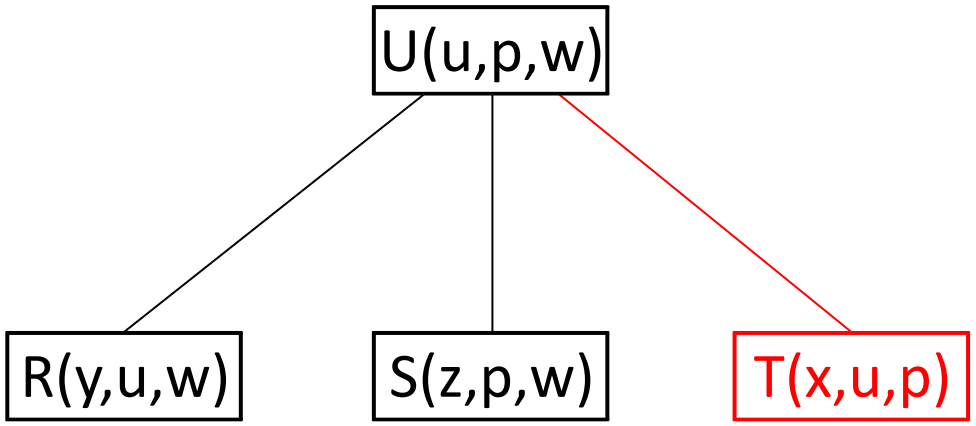
GYO reduction: Example 1

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

- GYO Ear removal
  - remove ears (= edges) greedily from hypergraph
    - 1. remove isolated nodes (also called singleton variables)
    - 2. remove consumed or empty edges (atoms)



Join tree

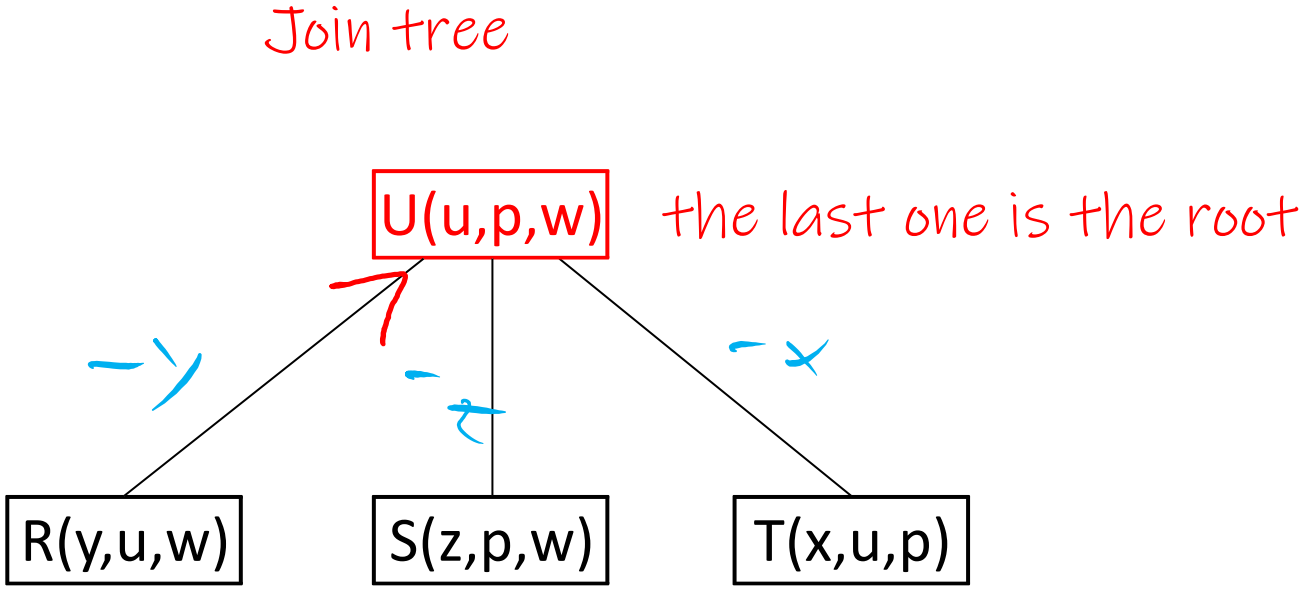


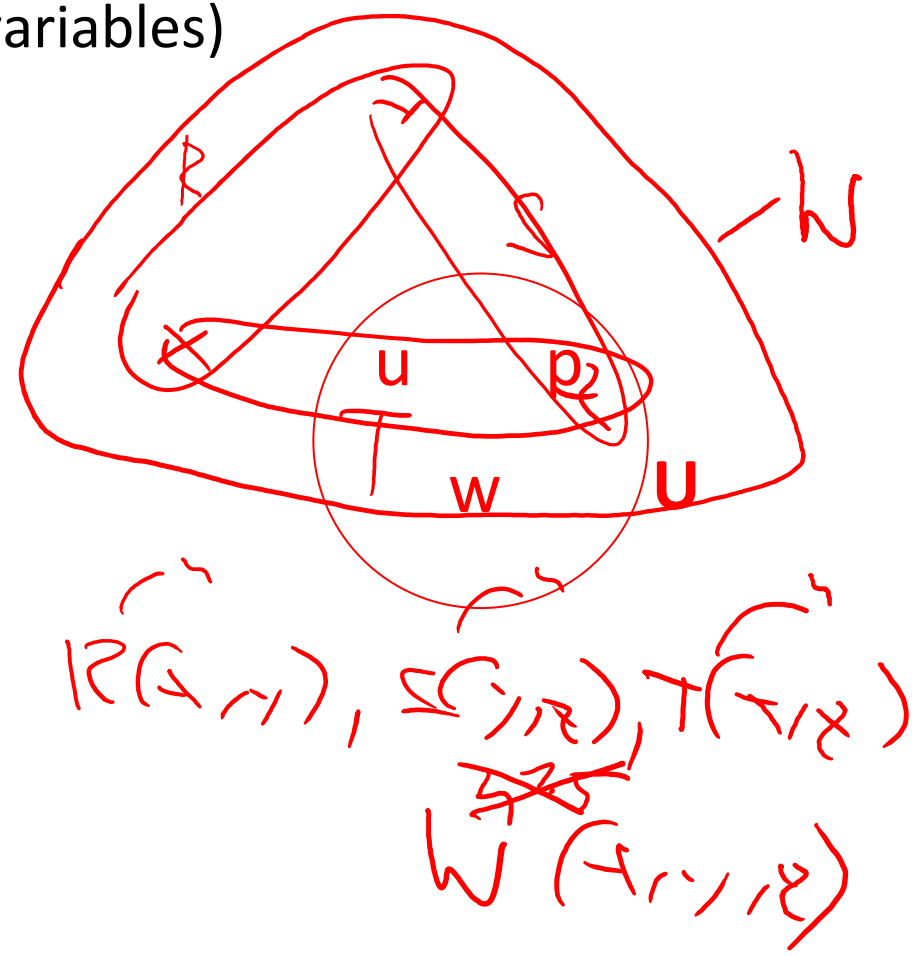
GYO reduction: Example 1

- GYO Ear removal

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

  - remove ears (= edges) greedily from hypergraph
    - 1. remove isolated nodes (also called singleton variables)
    - 2. remove consumed or empty edges (atoms)





GYO reduction: Example 2

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

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

Join tree

?

Query hypergraph

?