CompSci 516
Data Intensive Computing Systems

Lecture 22
Acyclic Joins
and
Worst Case Join Results

Instructor: Sudeepa Roy

Announcements

- Final exam according to the university exam schedule:
  - Monday, December 19, 9:00 am to 12 noon
- HW3 due tonight: 11/16
- Feedback on midterm project reports by tonight
  - you can schedule a meeting before the thanksgiving break
- Advanced topics covered for the next 2.5 lectures
- Project presentations and demos for the last 1.5 lectures
  - 10 projects
  - 10 mins each
- The most popular project as decided by your votes gets a prize!
  - You will give ratings to projects other than yours and the average will be taken
  - Unrelated to the grade of the project (given by the instructor)

Where are we now?

- ✓ Relational Model and Query Languages
  - ✓ SQL, RA, RC
  - ✓ Postgres (DBMS)
  - ▪ HW1
- ✓ DBMS Internals
  - ✓ Storage
  - ✓ Indexing
  - ✓ Query Evaluation
  - ✓ Operator Algorithms
  - ✓ External sort
  - ✓ Query Optimization
- ✓ Database Normalization

- ✓ Transactions
  - ✓ Basic concepts
  - ✓ Concurrency control
  - ✓ Recovery
- ✓ Query processing with multiple machines
  - ✓ Map-reduce and spark
  - ▪ HW2
  - ✓ Parallel DBMS
  - ✓ Distributed DBMS
- ✓ NOSQL
  - ▪ HW3
- ✓ Data warehouse and Cube
- ✓ Association Rule Mining
- ✓ Datalog

Next few lectures

- Overview of a few other research areas and topics in databases
  - lecture slides will be sufficient as reading material
  - additional reading material will be posted on the website

- Topics to be covered
  - Acyclic joins and new worst case join results (today)
  - Data integration
  - Schema matching and mapping
  - Data cleaning

Today's topics

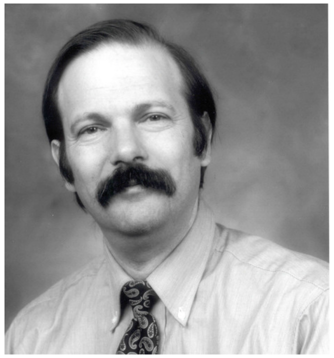
- Acyclic joins and query hypergraphs
  - (Mostly) using slides by Jeff Ullman and Georg Gottlob
- Worst case join results
  - Using slides by Ashwin Machanavajjhala
- These two topics will also give you an idea of research in database theory
  - and how they have impact on efficiency in database systems

Query Hypergraphs and Acyclic Joins

Based on slides by

Jeff Ullman and Georg Gottlob

Data and Query Complexity



- Inputs
  - Database D = {R1, …, Rk}
  - (Boolean) Query Q, size ~ k
  - # of tuples = n
  - Size of active domain |adom|
- Vardi’82 : complexity of answering if Q(D) is non-empty
- Data complexity
  - Fix query, k = constant, parameter = n
- Query or expression complexity
  - Fix D or n or |adom|
- Combined complexity
  - Both n and k are variables

Complexity of evaluating CQ

- Arbitrary CQ in Datalog notation
- Q() :- R1(**x1**), R2(**x2**), …, Rk(**xk**)

- e.g.
- Q() :- Sailors(sid, name, age), Boats(bid, `blue'), Reserve(sid, bid, `Monday')
- Q():- R(a, b, c), S(c, d), T(a, d)

- What is a trivial algorithm?

Complexity of evaluating CQ

- Arbitrary CQ Q() :- R1(**x1**), R2(**x2**), …, Rk(**xk**)
- What is a trivial algorithm?

- Iterate over all possible combinations of values of variables from their active domains
- Check if they belong to (satisfy) R1, …, Rk

Complexity ~ $O(n^k)$

Polynomial data complexity (n)

Exponential query complexity (k)

Complexity of evaluating CQ

- Arbitrary CQ Q() :- R1(**x1**), R2(**x2**), …, Rk(**xk**)
- What is a trivial algorithm?

- Iterate over all possible combinations of values of variables from their active domains
- Check if they belong to (satisfy) R1, …, Rk

Complexity ~ O(n^k)

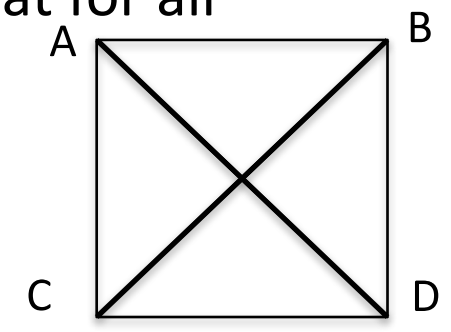
Polynomial data complexity (n)

Exponential query complexity (k)

Can we do better in terms of query complexity?

Complexity of evaluating CQ

- Evaluation of CQ is NP-hard in k [Chandra-Merlin'77]
- e.g. reduction from k-clique in G(V, E)
  - Find if there is a group of k vertices U such that for all u, v $\in$ U, the edge (u, v) $\subseteq$ E



a 4-clique

- $Q() :- \wedge_{1<i<j<k} E(x_i, x_j)$

Can we do better for some queries?

How about the following path query?

- Check if there is a path of length k-1

- $P^k() \text{ :- } R_1(x_1, x_2), R_2(x_2, x_3), \ldots., R_k(x_{k-1}, x_k)$

Note: the join size can be exponential

- Check if there is a path of length k-1

- $P^k() :- R_1(x_1, x_2), R_2(x_2, x_3), …., R_k(x_{k-1}, x_k)$

R_i

A_i	A_{i+1}
0	a
0	b
1	a
1	b
a	0
a	1
b	0
b	1

How about the following path query?

- Check if there is a path of length k-1
- $P^k() \text{ :- } R_1(x_1, x_2), R_2(x_2, x_3), \ldots., R_k(x_{k-1}, x_k)$

Give a poly-time (in k) algorithm for this query

Semi-join Opearator: $\ltimes$

- Recall semi-join from distributed databases (Lecture-17)
  - Project R to join columns and send to the site with S
  - Compute "reduction of S" by joining with the join columns
  - send back to the site with R and compute final join

- Instances
  - I of R, and
  - J of S

- $I \ltimes J = \pi_R (I \bowtie J)$

- $I \bowtie J = (I \ltimes J) \bowtie J = (J \ltimes I) \bowtie I$

Poly-time algorithm for path query

- $P^k() \text{ :- } R_1(x_1, x_2), R_2(x_2, x_3), \ldots., R_k(x_{k-1}, x_k)$

- R1’’ = Project R1 on x2
- R2’ = Semi-join R2 with R1’’
- R2’’ = Project R2’ on x3
- R3’ = Semi-join R3 with R1’’
- R3’’ = Project R3’ on x4
- ……

- Complexity:
  – suppose |Ri| = n
  – All intermediate relation size is at most n
  – Semi-join = O(n log n)
  – Running time = O(kn log n)

Acyclic query processing in poly-time generalizes this concept

Next (from Jeff Ullman's talk)

- Query hypergraph
- GYO reduction
- Acyclicity
- Four equivalent properties of acyclic queries
  1. GYO reduction is empty
  2. "Locally consistent" = "Globally consistent"
  3. A "full reducer" using semi-join exists
  4. Query has a "join tree"

Ron Fagin

Jeff Ullman

Phil Bernstein

Georg Gottlob

Ron Fagin and Acyclic Hypergraphs

Why Hypergraphs?

Interesting Properties

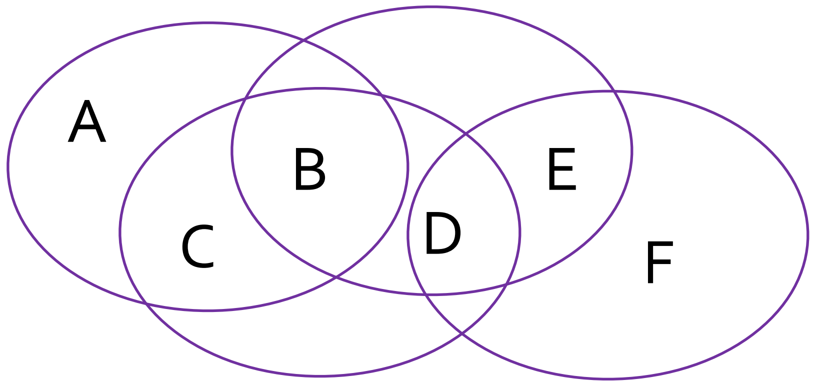
Fagin's Hierarchy

Jeffrey D. Ullman

Stanford University

Hypergraphs

- Nodes + *(hyper)edges* that are sets of any number of nodes.

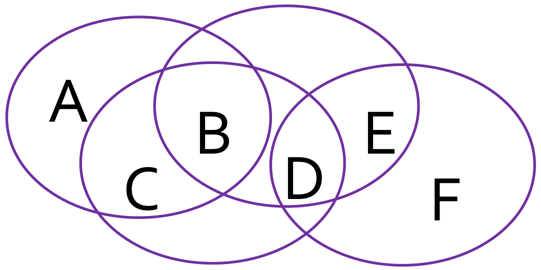


Hypergraphs as Schemas

- Nodes = attributes.
- Hyperedges = relation schemas.
- Hypergraph = database schema.

Hypergraphs as Schemas

- Nodes = attributes.
- Hyperedges = relation schemas.
- Hypergraph = database schema.

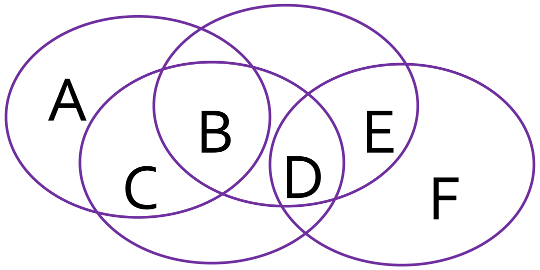


= {ABC, BCD, BDE, DEF}

Hypergraphs as Natural Joins

- Nodes = attributes.
- Edges = schemas of relations being joined.
  - Any equijoin can be so represented if we rename equated attributes from different relations.

Hypergraphs as Natural Joins

- Nodes = attributes.
- Edges = schemas of relations being joined.
  - Any equijoin can be so represented if we rename equated attributes from different relations.
- 

  = ABC $\bowtie$ BCD $\bowtie$ BDE $\bowtie$ DEF

Initial Study of Acyclic Hypergraphs for Database Systems

- Beeri, Fagin, Maier, Mendelzon, U, Yannakakis (STOC, 1981) looked at hypergraphs primarily as database schemas.

Initial Study of Acyclic Hypergraphs for Database Systems

- Beeri, Fagin, Maier, Mendelzon, U, Yannakakis (STOC, 1981) looked at hypergraphs primarily as database schemas.
- At that time, the “universal-relation wars” were raging.
  - Could you ask queries about attributes only and allow the system to figure out the proper join to connect these attributes?

Initial Study of Acyclic Hypergraphs for Database Systems

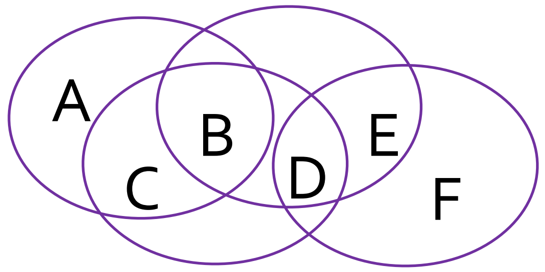
- Beeri, Fagin, Maier, Mendelzon, U, Yannakakis (STOC, 1981) looked at hypergraphs primarily as database schemas.
- At that time, the "universal-relation wars" were raging.
  - Could you ask queries about attributes only and allow the system to figure out the proper join to connect these attributes?
- Identified a class of schemas ("*acyclic*") with certain properties that made sense as a universal relation.

The GYO Test for Acyclicity

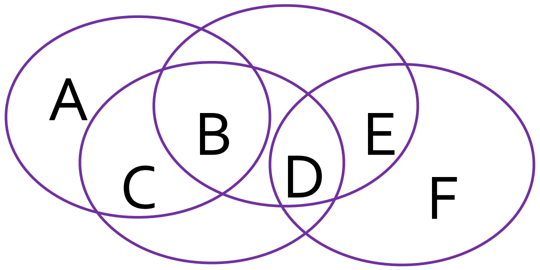
- It turns out there is a simple way to tell whether a hypergraph is acyclic, so we won’t bother with the original definition.
- Due to Graham and Yu-Oszoyoglu independently.
- “Reduce” the hypergraph using the following two rules:
  - Eliminate a node in only one hyperedge.
  - Eliminate a hyperedge contained in another.
- If you get down to one empty edge, then the hypergraph is acyclic.

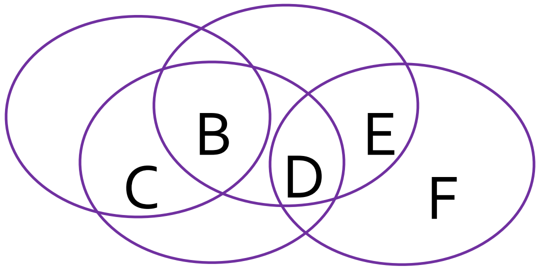
Example: GYO Reduction

Example: GYO Reduction

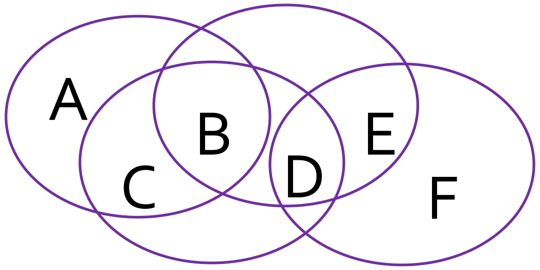


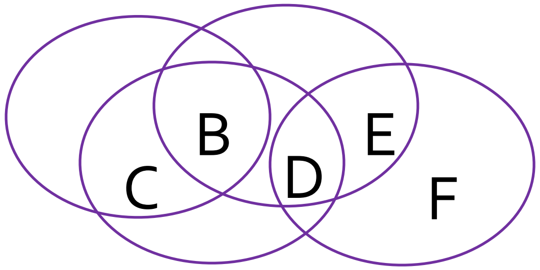
Example: GYO Reduction

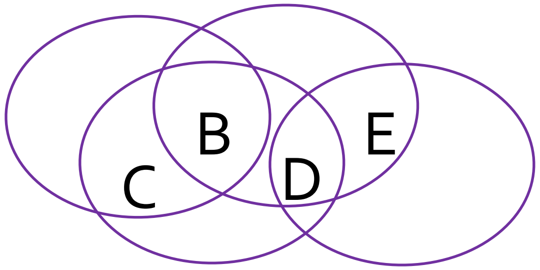




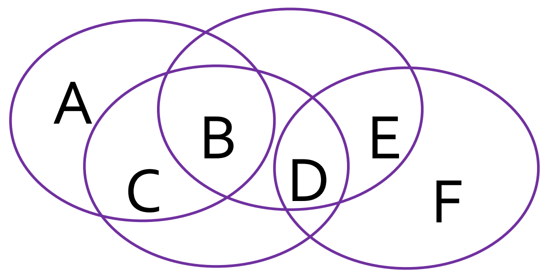
Example: GYO Reduction

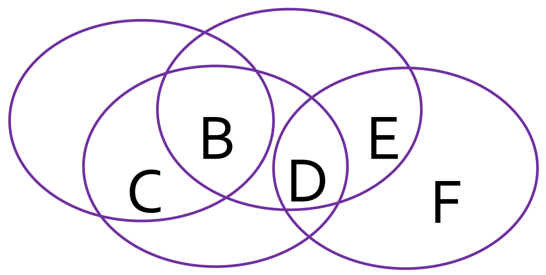


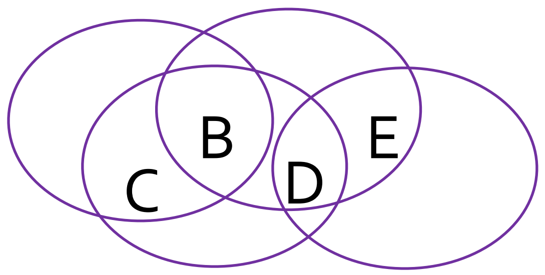


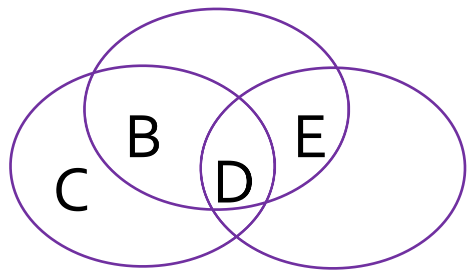


Example: GYO Reduction

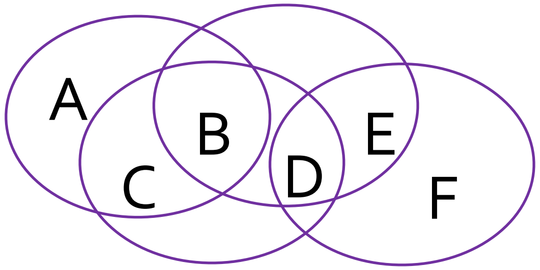


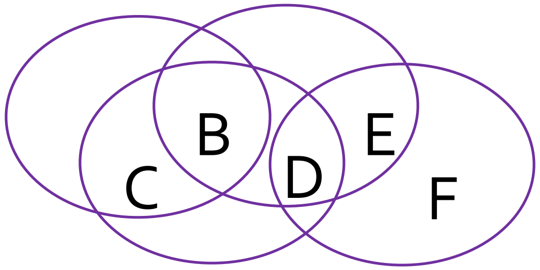


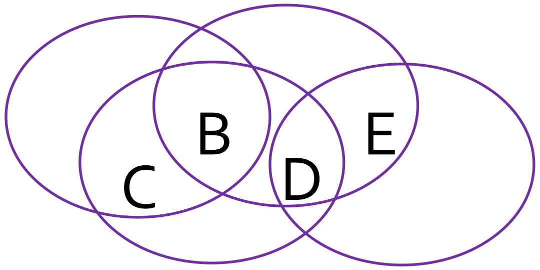


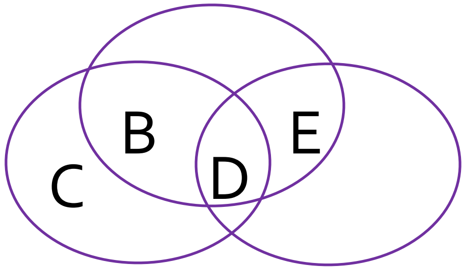


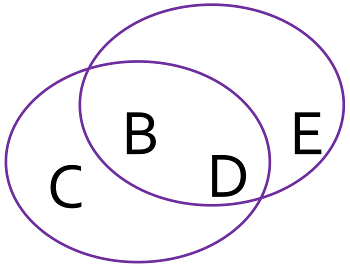
Example: GYO Reduction



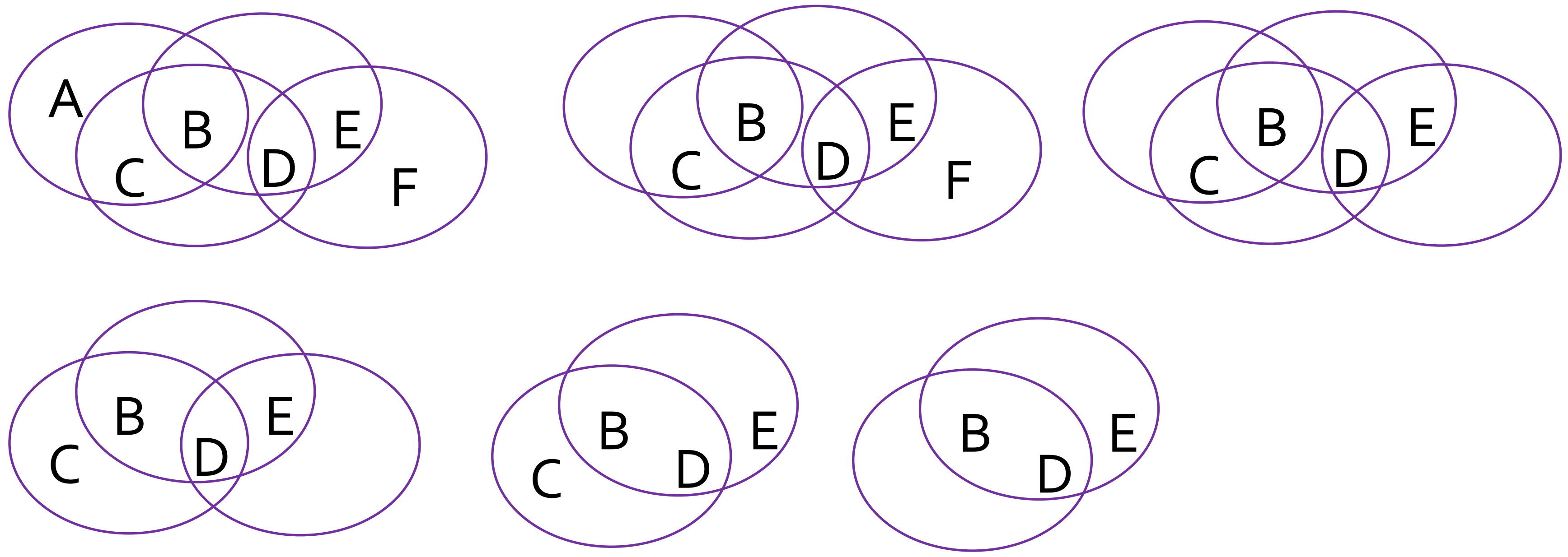




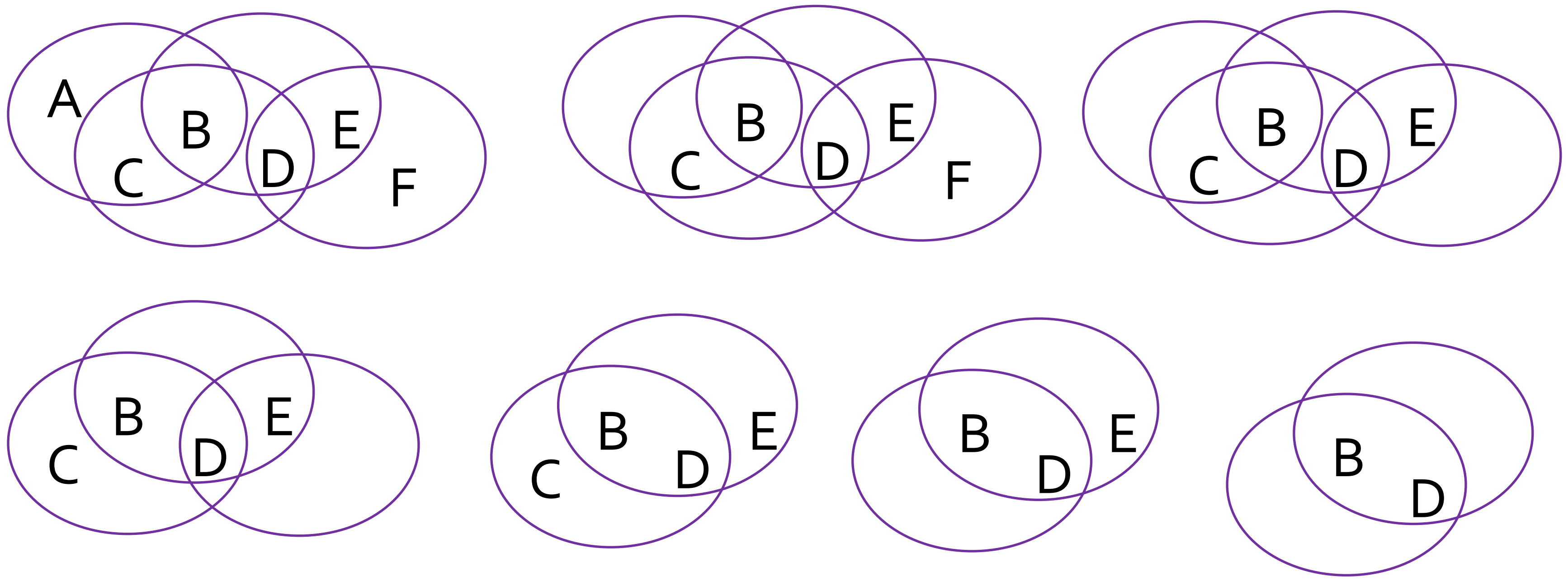




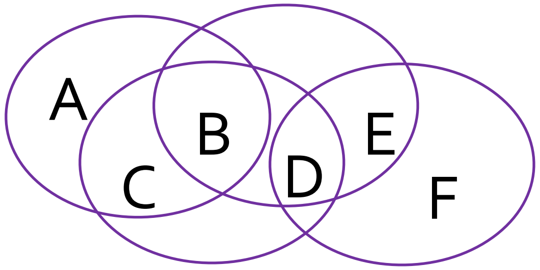
Example: GYO Reduction

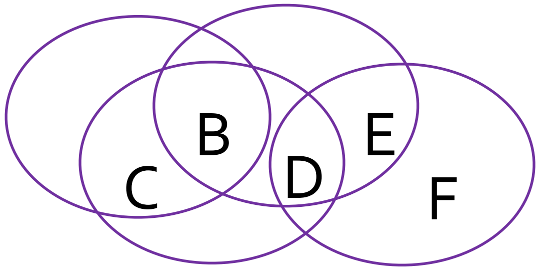


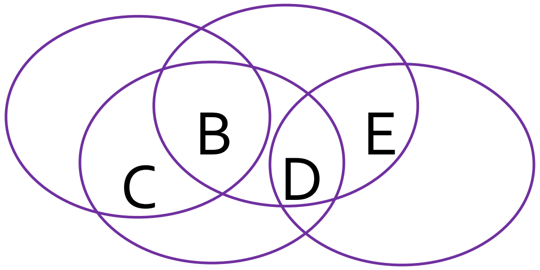
Example: GYO Reduction

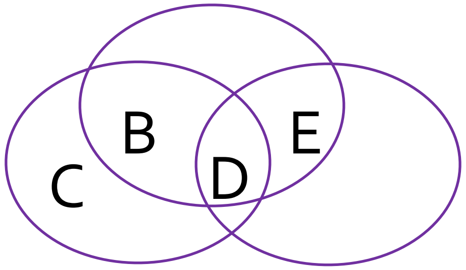


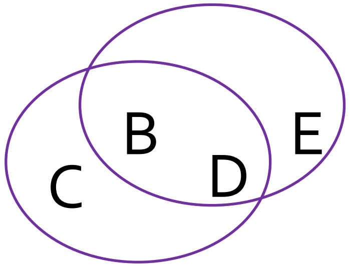
Example: GYO Reduction

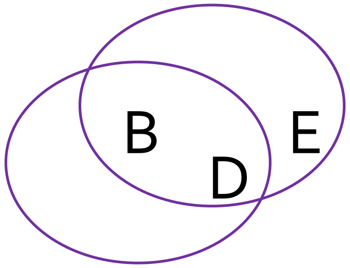


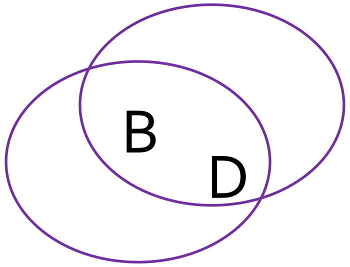


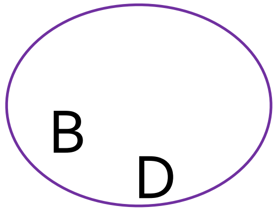




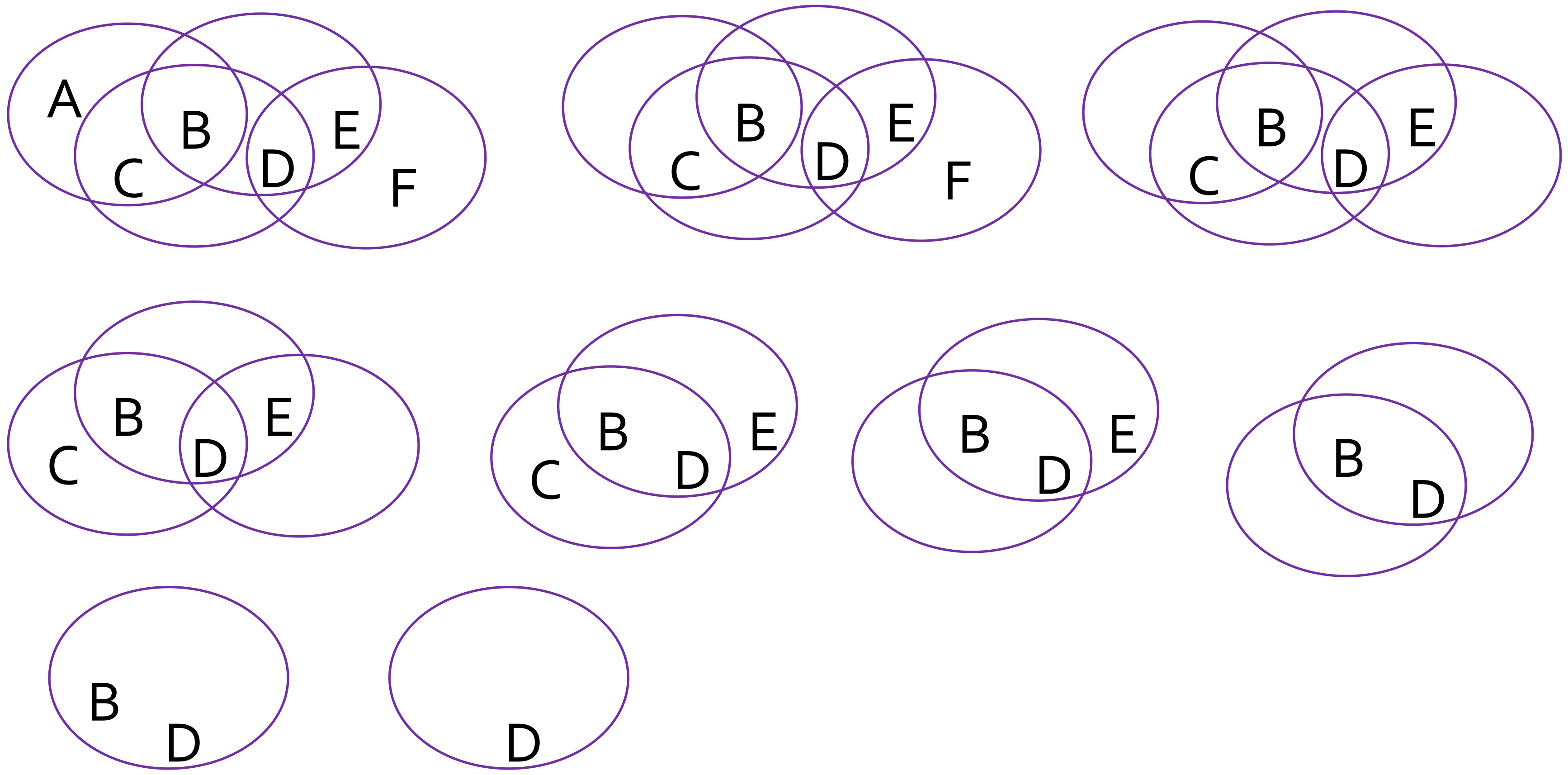




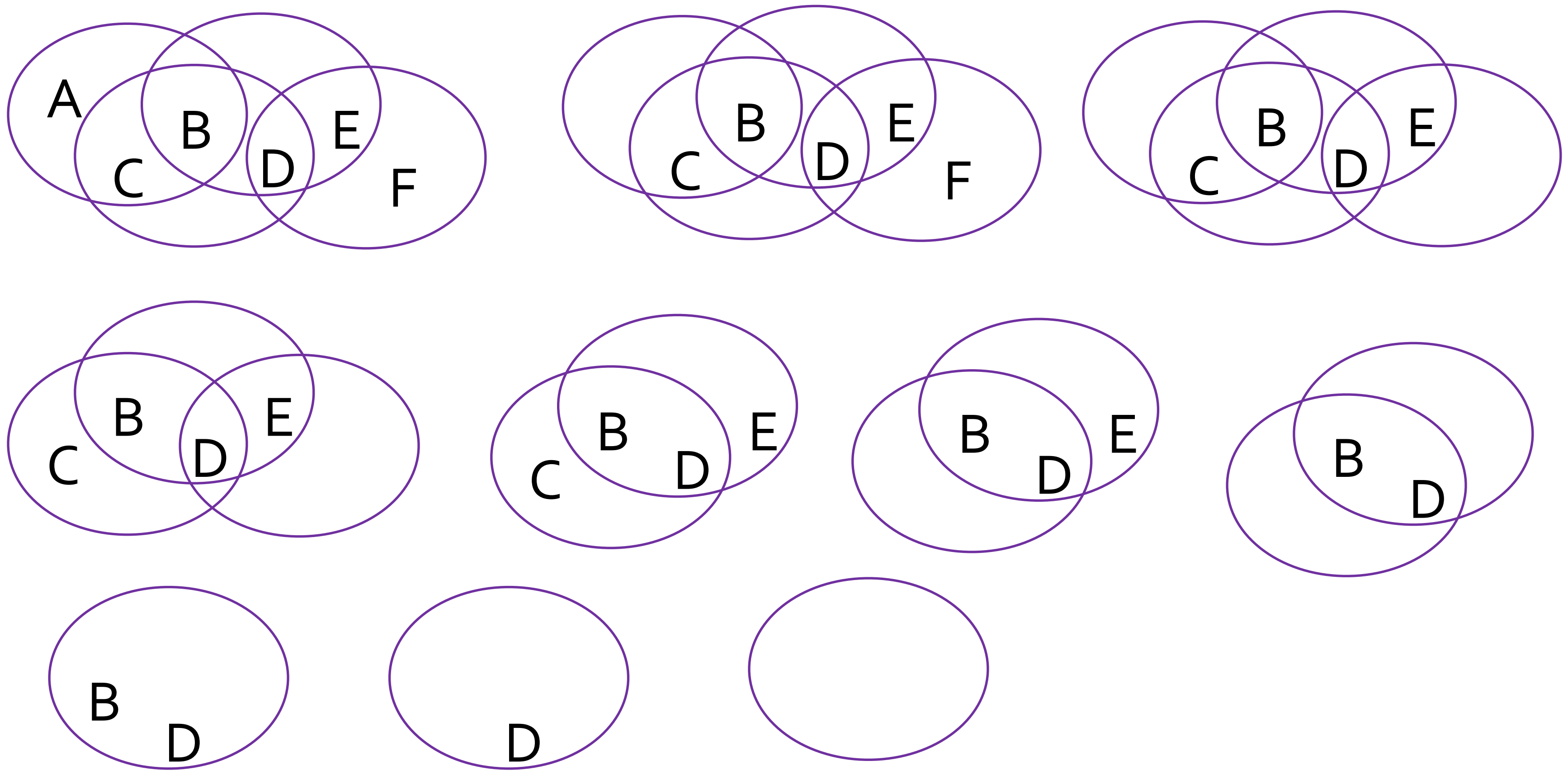




Example: GYO Reduction



Example: GYO Reduction



Semijoin Reductions

- Previously, Phil Bernstein and his students Chiu, Goodman, and Shmueli had looked at a seemingly unrelated question: when does a join have a *full reducer*?
  - = finite sequence of semijoins that is guaranteed to eliminate from the relations all tuples that dangle in the complete join.

Local and Global Consistency

- Schema = $R_1, R_2, \ldots, R_k$
- Instances = $I_1, I_2, \ldots, I_k$

- Locally consistent
  - every tuple participates in pairwise joins
  - for all j, k
  - $\pi_{Rj}\ (I_j \bowtie I_k) = I_j$

- Globally consistent
  - every tuple participates in full join
  - for all j
  - $\pi_{Rj}\ (I_1 \bowtie I_2 \bowtie I_2 \bowtie \ldots \bowtie I_k) = I_j$

Local and Global Consistency

- A related formulation: when does *local consistency*
  - = the join of any two relations has no dangling tuples
- imply *global consistency*
  - = there are no dangling tuples in any relation when the join of all the relations is taken.
- It turns out "exists a full reducer" = "local consistency implies global consistency" = "acyclic."

Example: Local/Global Consistency

A	B
0	1
3	4
6	7

B	C
1	2
4	5
7	8

C	A
2	3
5	6
8	0

These three relations are locally consistent.
But the join of all three relations is empty.
Hence not globally consistent.

Example: Semijoin Reduction

A	B
0	1
3	4
6	7

B	C
1	2
4	5
7	8

C	A
2	3
5	6
8	9

Notice the change

Now, semijoin reduction will make each relation empty.
But the number of steps needed depends on the number of tuples.

1. AB $\ltimes$ CA eliminates only (0,1).
2. Then BC $\ltimes$ AB eliminates only (1,2).
3. And so on…

Monotone Joins

- A join of two relations is *monotone* if it has no dangling tuples.

Monotone Joins

- A join of two relations is *monotone* if it has no dangling tuples.
- Important consequence: the output of a monotone join is at least as large each of its arguments.
  - If implemented properly, the time taken by the join is proportional to input size + output size.

Monotone Joins

- A join of two relations is *monotone* if it has no dangling tuples.
- Important consequence: the output of a monotone join is at least as large each of its arguments.
  - If implemented properly, the time taken by the join is proportional to input size + output size.
- Note: "local consistency" = "joins of two database relations are monotone," but "monotone" applies to intermediate joins also.

Bernstein et al. View of Acyclicity

- This line of research had a very different view of the condition under which full reducers exist (and under which local consistency = global consistency).

Bernstein et al. View of Acyclicity

- This line of research had a very different view of the condition under which full reducers exist (and under which local consistency = global consistency).
- If and only if you can build a tree with:

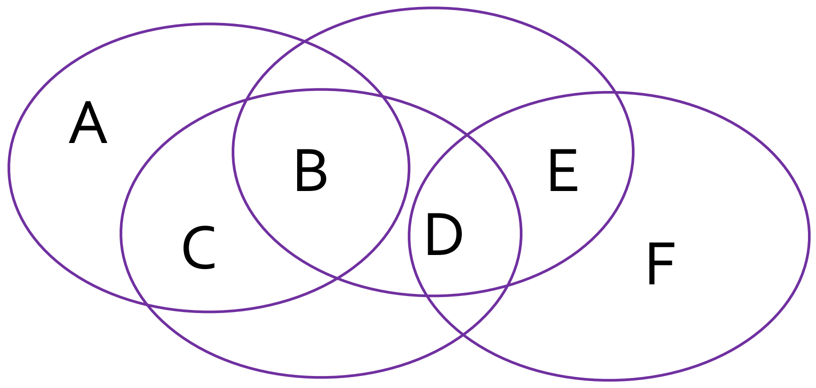
Bernstein et al. View of Acyclicity

- This line of research had a very different view of the condition under which full reducers exist (and under which local consistency = global consistency).
- If and only if you can build a tree with:
  - Nodes = relation schemas.

Bernstein et al. View of Acyclicity

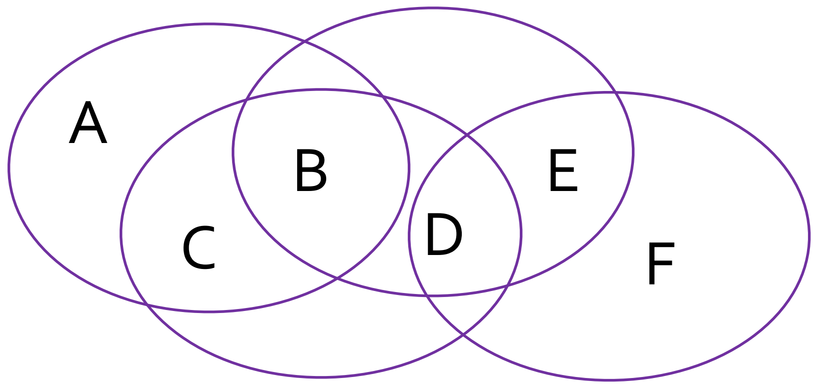
- This line of research had a very different view of the condition under which full reducers exist (and under which local consistency = global consistency).
- If and only if you can build a tree with:
  - Nodes = relation schemas.
  - For every attribute, the set of nodes containing that attribute is connected.

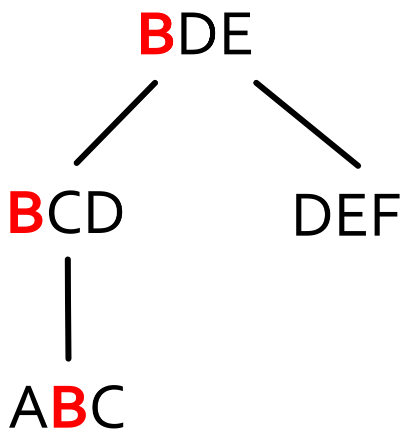
Example: Tree View of Acyclicity



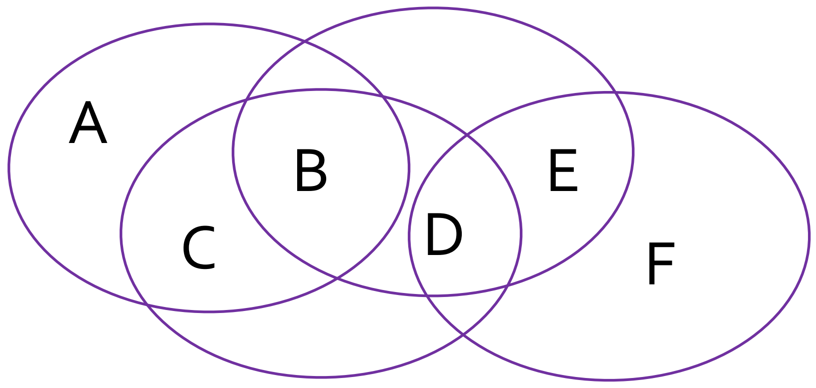
$$\begin{tikzcd}
 & BDE \arrow[dl, no head] \arrow[dr, no head] & \\
BCD \arrow[d, no head] & & DEF \\
ABC & &
\end{tikzcd}$$

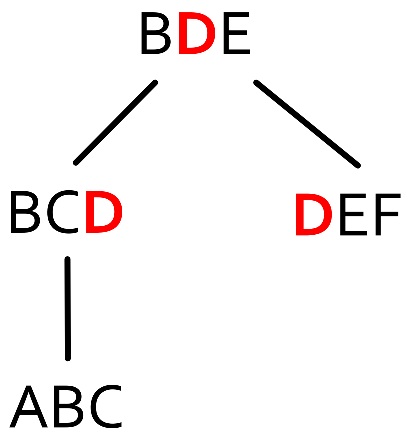
Example: Tree View of Acyclicity



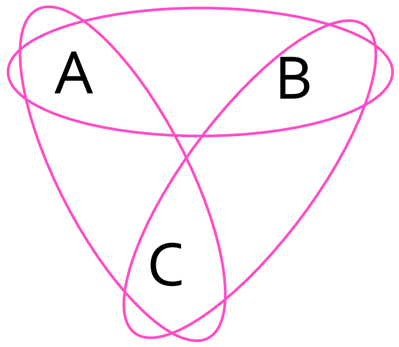


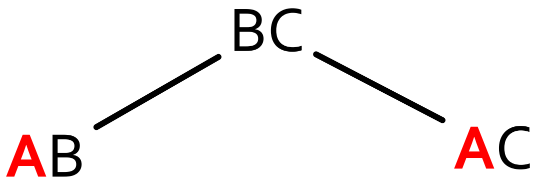
Example: Tree View of Acyclicity





Example: A Cyclic Join





By symmetry, all trees look like this.
Notice A is at disconnected nodes.

Theorem

- From Beeri, Fagin, Maier, and Yannakakis (J. ACM, 1983).

Theorem

- From Beeri, Fagin, Maier, and Yannakakis (J. ACM, 1983).
- A hypergraph is acyclic if and only if its hyperedges form a tree whose nodes containing any given attribute are connected.

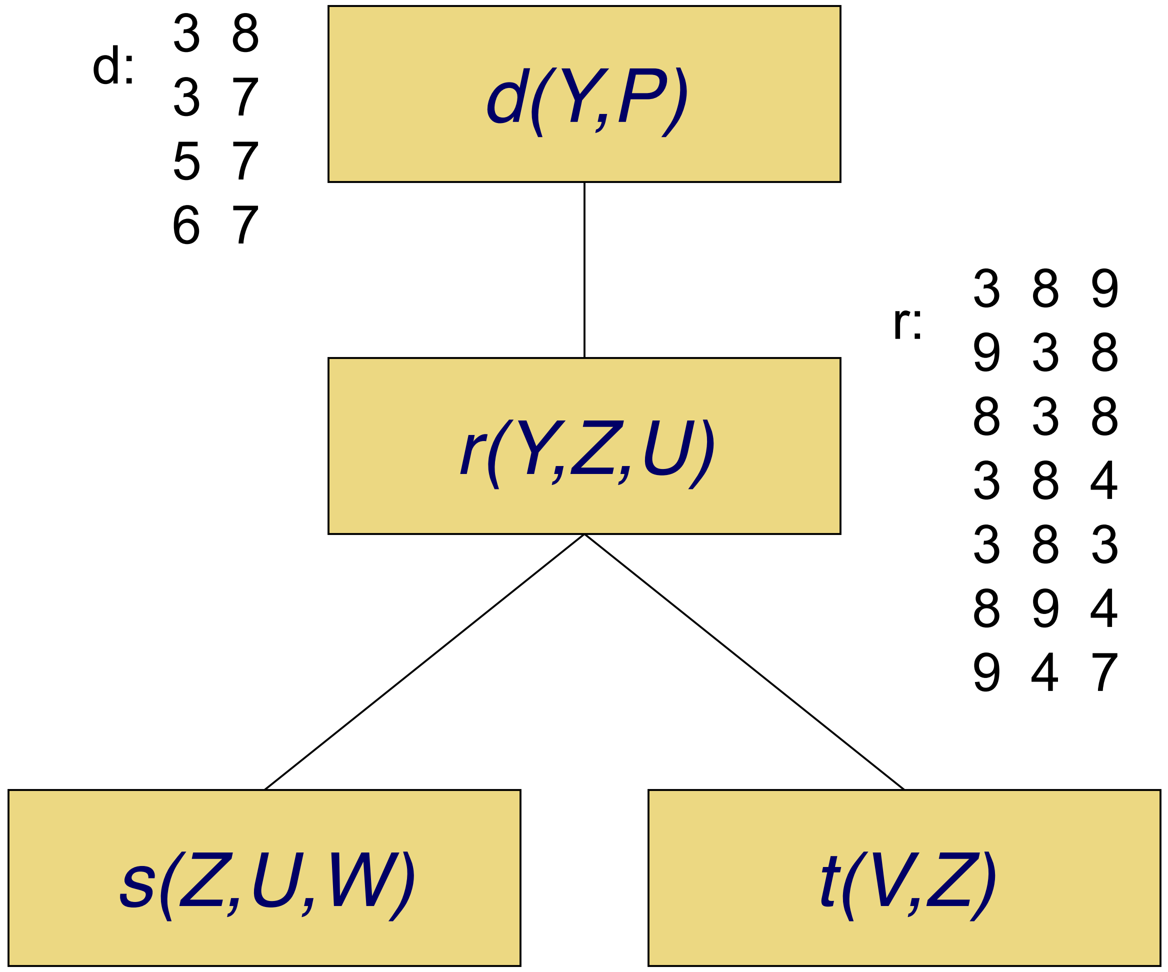
Theorem

- From Beeri, Fagin, Maier, and Yannakakis (J. ACM, 1983).
- A hypergraph is acyclic if and only if its hyperedges form a tree whose nodes containing any given attribute are connected.
- Therefore, acyclic hypergraphs, and only acyclic hypergraphs, have:
  1. Full reducers.
  2. Local consistency = global consistency.
  3. Local consistency => monotone join sequences guaranteed to exist.

How does query acyclicity help obtain an efficient join algorithm?

Example from Georg Gottlob’s talk

How are ACQs evaluated according to Yannakakis‘Algorithm?



d:

3	8
3	7
5	7
6	7

r:

3	8	9
9	3	8
8	3	8
3	8	4
3	8	3
8	9	4
9	4	7

s:

3	8	9
9	3	8
8	3	8
3	8	4
3	8	3
8	9	4
9	4	7

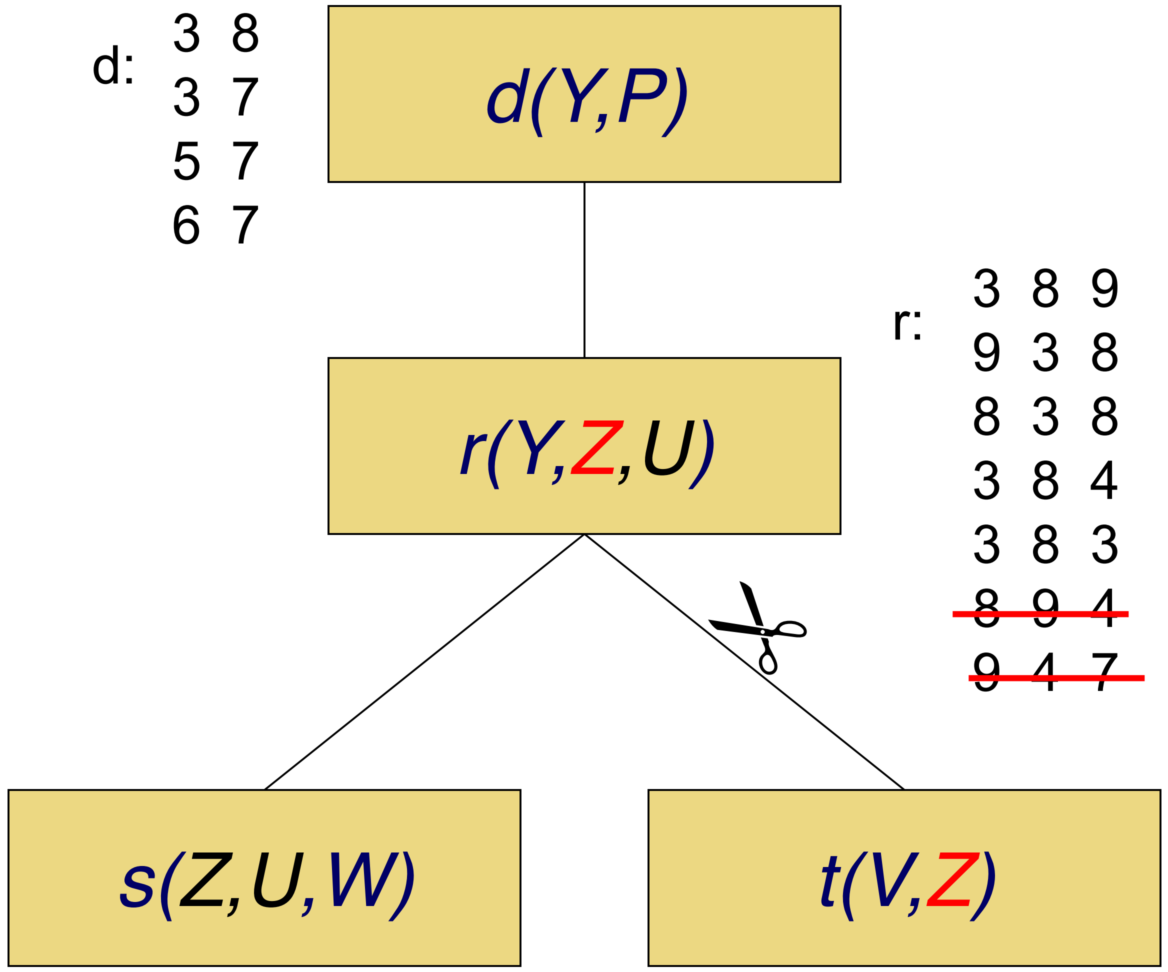
t:

9	8
9	3
9	5

$$O(|Q| \times |r_{max}| \times \log |r_{max}|)$$

d:

3	8
3	7
5	7
6	7



r:

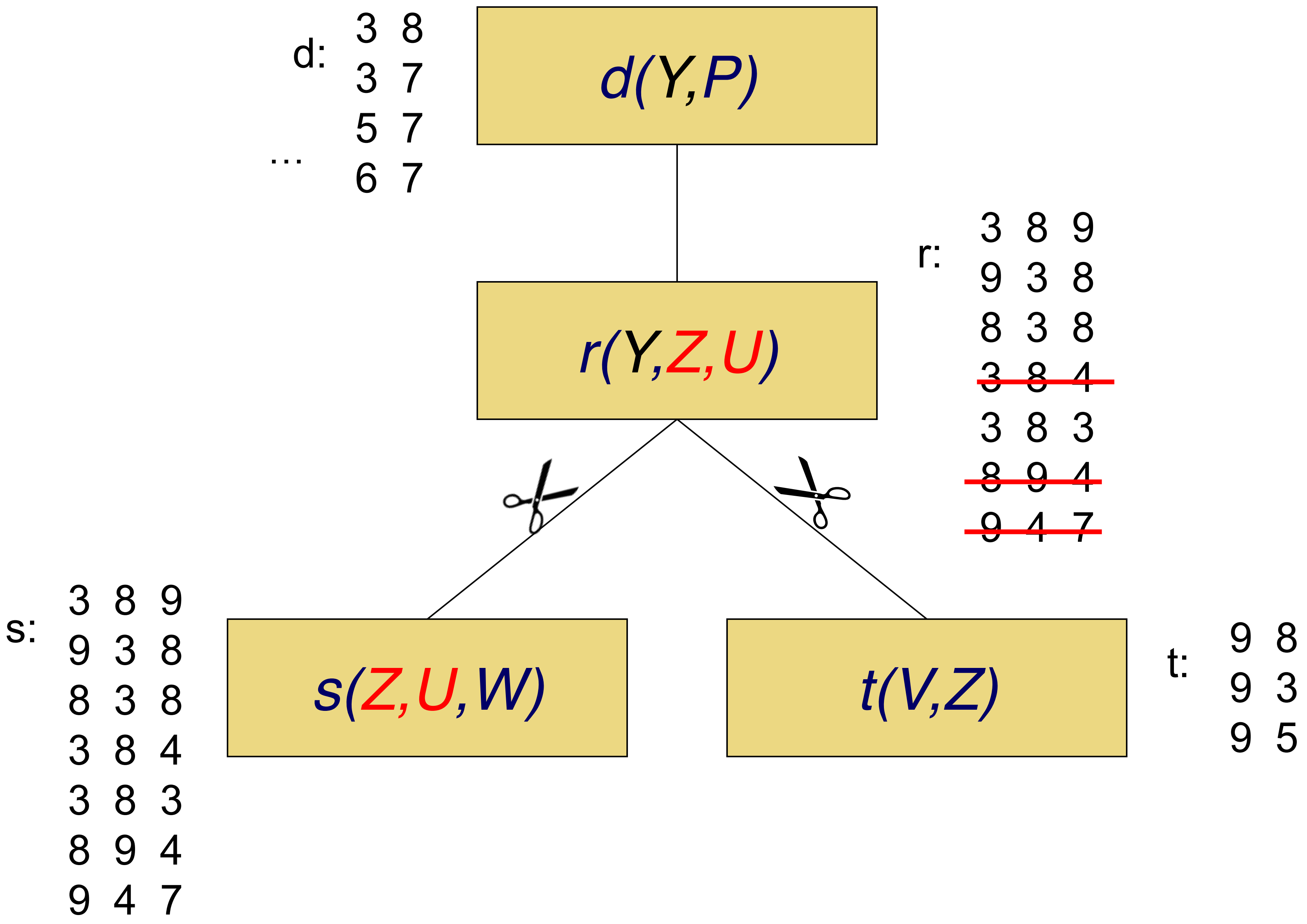
3	8	9
9	3	8
8	3	8
3	8	4
3	8	3
~~8~~	~~9~~	~~4~~
~~9~~	~~4~~	~~7~~

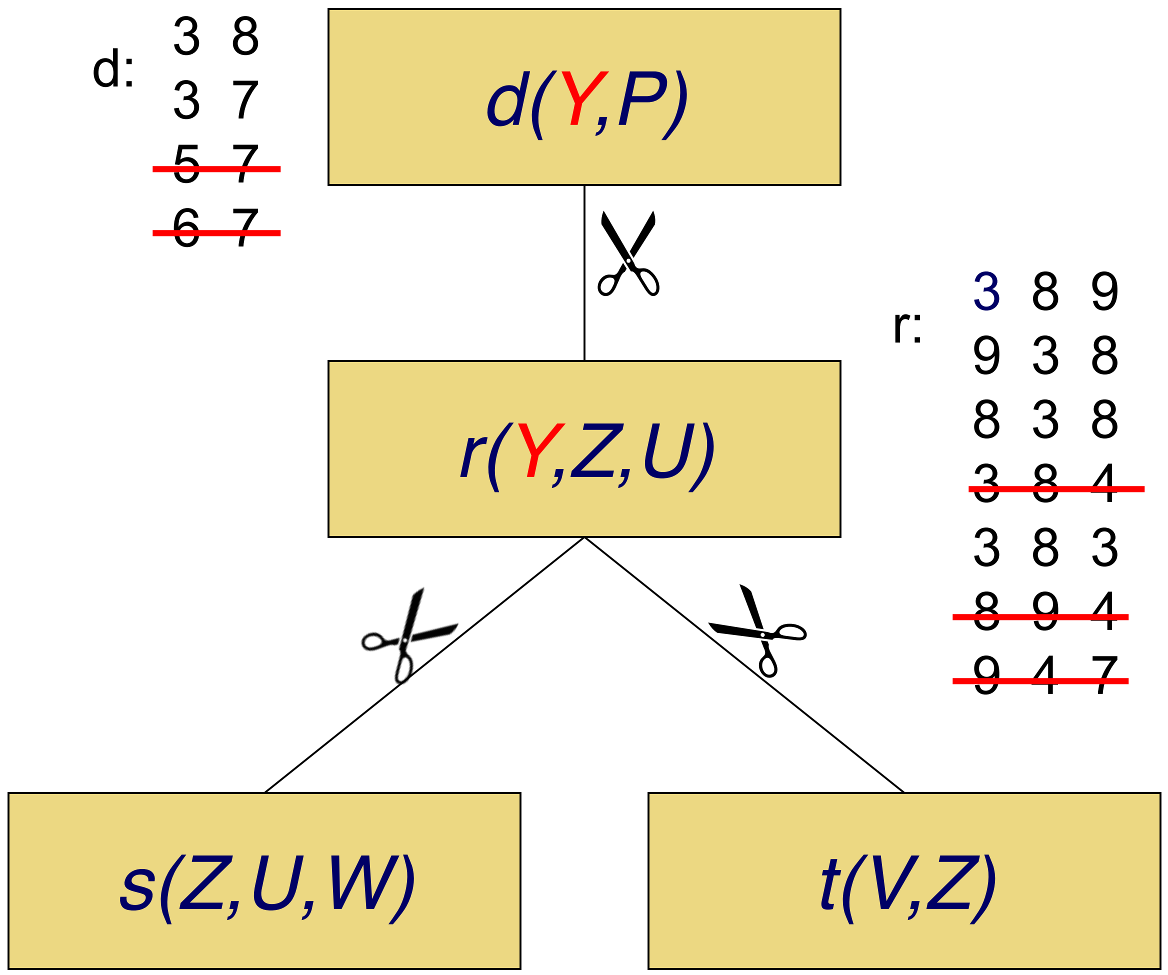
s:

3	8	9
9	3	8
8	3	8
3	8	4
3	8	3
8	9	4
9	4	7

t:

9	8
9	3
9	5


d(Y,P)
r(Y,Z,U)
s(Z,U,W)
t(V,Z)
d:
3 8
3 7
5 7
…
6 7
r:
3 8 9
9 3 8
8 3 8
~~3 8 4~~
3 8 3
~~8 9 4~~
~~9 4 7~~
s:
3 8 9
9 3 8
8 3 8
3 8 4
3 8 3
8 9 4
9 4 7
t:
9 8
9 3
9 5



d:

3	8
3	7
~~5~~	~~7~~
~~6~~	~~7~~

r:

3	8	9
9	3	8
8	3	8
~~3~~	~~8~~	~~4~~
3	8	3
~~8~~	~~9~~	~~4~~
~~9~~	~~4~~	~~7~~

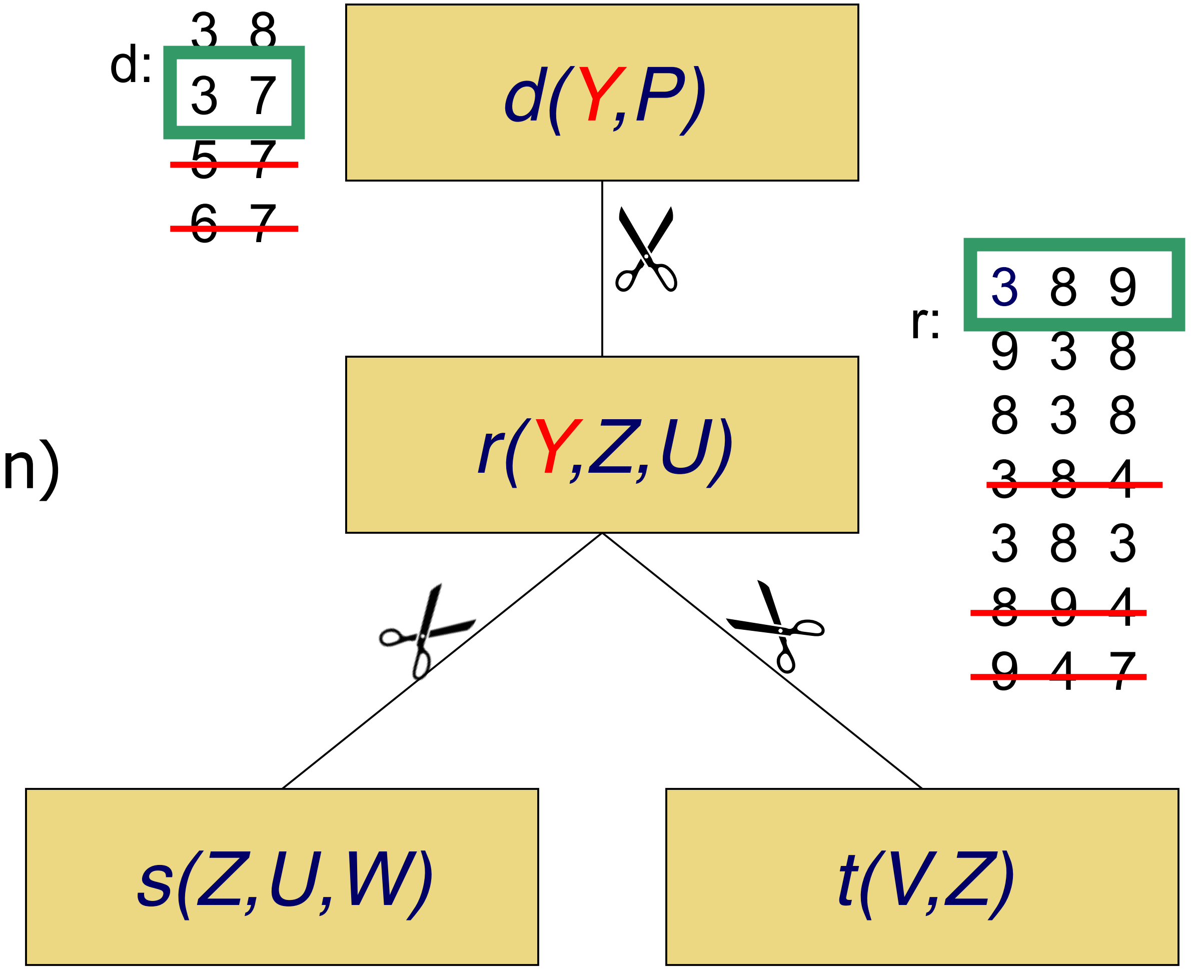
s:

3	8	9
9	3	8
8	3	8
3	8	4
3	8	3
8	9	4
9	4	7

t:

9	8
9	3
9	5

$O(|Q| \times |r_{max}| \times \log |r_{max}|)$



d:

3	8
$\boxed{3\ \ 7}$	
~~5~~	~~7~~
~~6~~	~~7~~

r:

$\boxed{3\ \ 8\ \ 9}$		
9	3	8
8	3	8
~~3~~	~~8~~	~~4~~
3	8	3
~~8~~	~~9~~	~~4~~
~~9~~	~~4~~	~~7~~

O(n^2log n)

s:

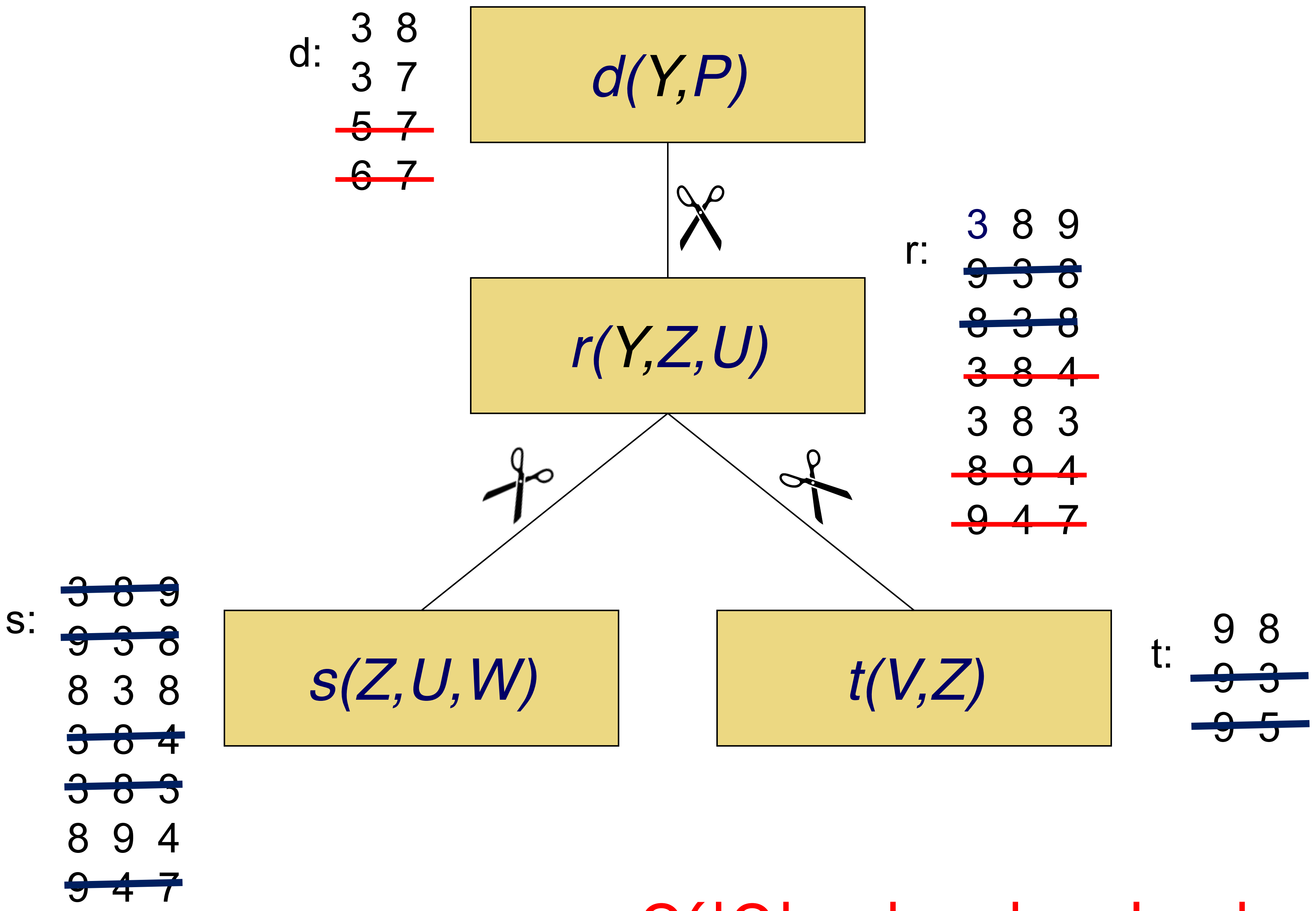
3	8	9
9	3	8
8	3	8
3	8	4
3	8	3
$\boxed{8\ \ 9\ \ 4}$		
9	4	7

t:

$\boxed{9\ \ 8}$	
9	3
9	5

A solution: Y=3, P=7, Z=8, U=9, W=4, V=9

To obtain fully reduced relations, perform semijoins downwards



$O(|Q| \times |r_{max}| \times \log |r_{max}|)$

Yannakakis'[84] Algorithm for Acyclic Boolean CQ

1. Compute a join-tree from GYO decomposition
   - Need the "formal definition" of GYO reduction
   - an edge f can be removed if there exists another edge f' as "witness" such that no vertex of f-f' is in any other edge
   - == f can be partitioned into vertices in no other edges + vertices contained in f'
   - f' becomes the parent of f in the join-tree
2. Compute semi-join at each parent node bottom-up for each edge to a child
3. Query is true if and only if the root relation is non-empty

Full Reducer from Yannakakis's algorithm

- To obtain fully reduced relations:
  - After reaching the root, perform semi-joins downwards

- Algorithm to obtain full reducers using the join-tree:
  - if (f, f') is an edge in the tree where f is the child
    - Add f' = f' $\ltimes$ f
    - Recursively obtain reducer for the tree removing f

      (here we have a globally-consistent state)
    - Add f := f $\ltimes$ f'
  - This is done for distributed semi-join!

Full Reducer for Path Query

- $P^k() \text{ :- } R_1(x_1, x_2), R_2(x_2, x_3), \ldots., R_k(x_{k-1}, x_k)$

- $R_2 := R_2 \ltimes R_1$
- $R_3 := R_3 \ltimes R_2$
- …..
- $R_k := R_k \ltimes R_{k-1}$

Enough for query evaluation

- $R_{k-1} := R_{k-1} \ltimes R_k$
- …..
- $R_2 := R_2 \ltimes R_3$
- $R_1 := R_1 \ltimes R_2$

Extension of Yannakakis' algorithm for non-Boolean CQs

- Polynomial in input size + output size

1. Use a full reducer (like before)
2. Join in any order
3. In the join phase, project out all unnecessary attributes
   - while joining P $\bowtie$ C : P parent and C child
   - project out all attributes in C that are not in the final projection

- No globally dangling tuples, so join can only increase in size at each step, each intermediate result is polynomial in the output size
- Unlike cyclic join (add a cycle in our first example)

Other Notions of Acyclicity by Fagin'83

- Notion of CQ acyclicity through GYO reduction = **α**-acyclic
- Other notions in Fagin'83: γ-acyclicicity and β-acyclicity
  - Also Berge-acyclicity (standard notion of acyclicity in graphs generalized to hypergraphs)
  - these are not covered in class

References

- Ron Fagin event talk in PODS'16 by Jeff Ullman
- Jeff Ullman's course notes: http://infolab.stanford.edu/~ullman/cs345-notes.html
- Gems of PODS'16 talk by Georg Gottlob
  - Gottlob et al. generalized the notion of acyclicity to "generalized hyper-tree width"
  - Talks about incorporating hypertree in PostGres and challenges
  - Many other applications and genralization
  - Both slides and an article are available online
- Check out https://databasetheory.org
  - Links to both talks can be found here (and more!)
- Alice Book (Foundations of Databases – Abiteboul, Hull, Vianu) Chapter 6.4

Topic#2: Worse-case join algorirgms

See slides by Ashwin Machanavajjhala
on the course website

(full slide deck can be found from this link:
https://www.cs.duke.edu/courses/fall15/compsci590.4/slides/compsci590.04_fall15_lec19.pdf
which includes lower bound results – not covered in this class)