Updated 3/29/2022

Topic 3: Efficient query evaluation
Unit 1: Acyclic query evaluation (continued)
Lecture 19

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp22)

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

3/29/2022

Pre-class conversations

- Current topic: trees vs cycles
- Scribes, projects, keep on commenting on slides

- Today:
  - Full Semi-join reducers
  - Cycles: tree decompositions, Linear Programming Duality

Algorithmic Problems in First-Order Logic

Truth-Evaluation Problem: Given a first-order formula $\varphi(x_1, \ldots, x_k)$, a graph $\mathbf{G}$, and a binding α, does $\mathbf{G} \models_\alpha \varphi(x_1, \ldots, x_k)$?

Satisfiability Problem: Given a first-order formula ψ, *is there* a graph $\mathbf{G}$ and binding α, such that $\mathbf{G} \models_\alpha \psi$?

Contrast:

- Satisfiability is *undecidable* [Church, Turing 1936]
- Truth evaluation, which is query evaluation, is *decidable*.

Source: Moshe Y. Vardi. "A Logical Revolution", 2013. https://www.cs.rice.edu/~vardi/comp409/logicrevo13.pdf , https://www.youtube.com/watch?v=7pu-ZJddxJQ&t=59m20s
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

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

Semijoin Reducer

$$Q(x,y,z) = R(x,y) \bowtie S(y,z) \bowtie T(z,w)$$

DEFINITION: A full semijoin reducer (semijoin program) is:

a sequence of semijoins on the join tree

$$R'(x,y) = R(y,z) \ltimes \ldots$$
$$S'(y,z) = S(y,z) \ltimes \ldots$$
$$\ldots$$

s.t. there no more "dangling tuples" in the reduced relations

Then you can rewrite the query over the reduced relations

$$Q(x,y,z) = R'(x,y) \bowtie S'(y,z) \bowtie T'(z,w)$$

Semijoin Reducer

$$Q(x,y,z) = R(x,y) \bowtie S(y,z) \bowtie T(z,w)$$

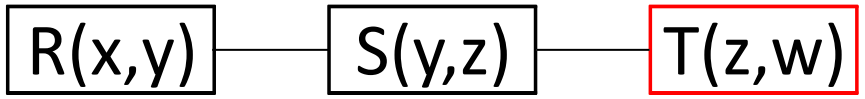
A full reducer is

?

Semijoin Reducer

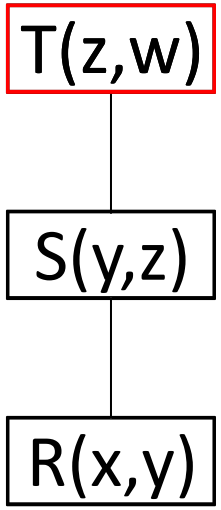
$$Q(x,y,z) = R(x,y) \bowtie S(y,z) \bowtie T(z,w)$$

A full reducer is



?

1. Find a join tree

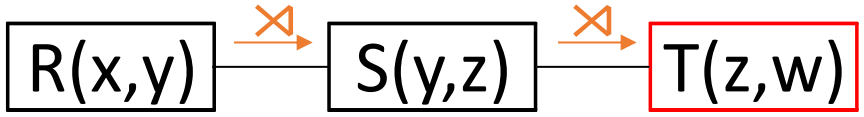


2. pick a root

Semijoin Reducer

$$Q(x,y,z) = R(x,y) \bowtie S(y,z) \bowtie T(z,w)$$

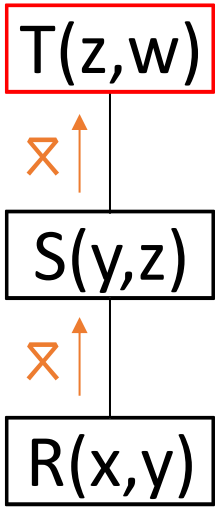
A full reducer is



3. collect at root (= bottom-up)

$$S_1(y,z) = S(y,z) \ltimes R(x,y)$$
$$T_1(z,y) = T(z,y) \ltimes S_1(y,z)$$

1. Find a join tree



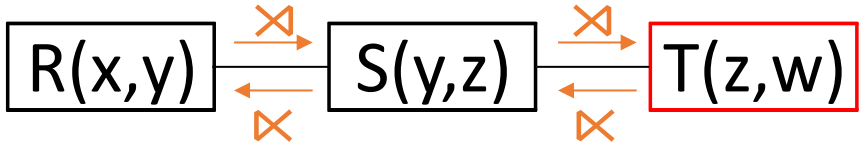
2. pick a root

3. up

Semijoin Reducer

$$Q(x,y,z) = R(x,y) \bowtie S(y,z) \bowtie T(z,w)$$

A full reducer is



3. collect at root (= bottom-up)

$$S_1(y,z) = S(y,z) \ltimes R(x,y)$$

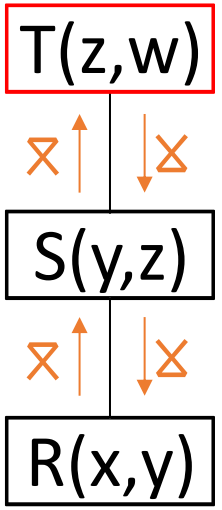
4. distribute to leaves (= top-down)

$$T_1(z,y) = T(z,y) \ltimes S_1(y,z)$$
$$S_2(z,y) = S_1(y,z) \ltimes T_1(z,y)$$
$$R_1(x,y) = R(x,y) \ltimes S_2(y,z)$$

5. The rewritten query is

?

1. Find a join tree



2. pick a root

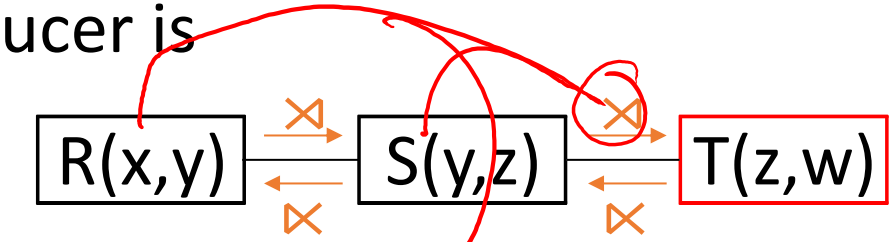
3. up

4. down

Semijoin Reducer

$$Q(x,y,z) = R(x,y) \bowtie S(y,z) \bowtie T(z,w)$$

A full reducer is



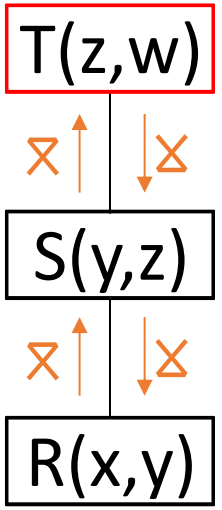
3. collect at root (= bottom-up)

$$S_1(y,z) = S(y,z) \ltimes R(x,y)$$

4. distribute to leaves (= top-down)

$$T_1(z,y) = T(z,y) \ltimes S_1(y,z)$$
$$S_2(z,y) = S_1(y,z) \ltimes T_1(z,y)$$
$$R_1(x,y) = R(x,y) \ltimes S_2(y,z)$$

1. Find a join tree



2. pick a root

3. up

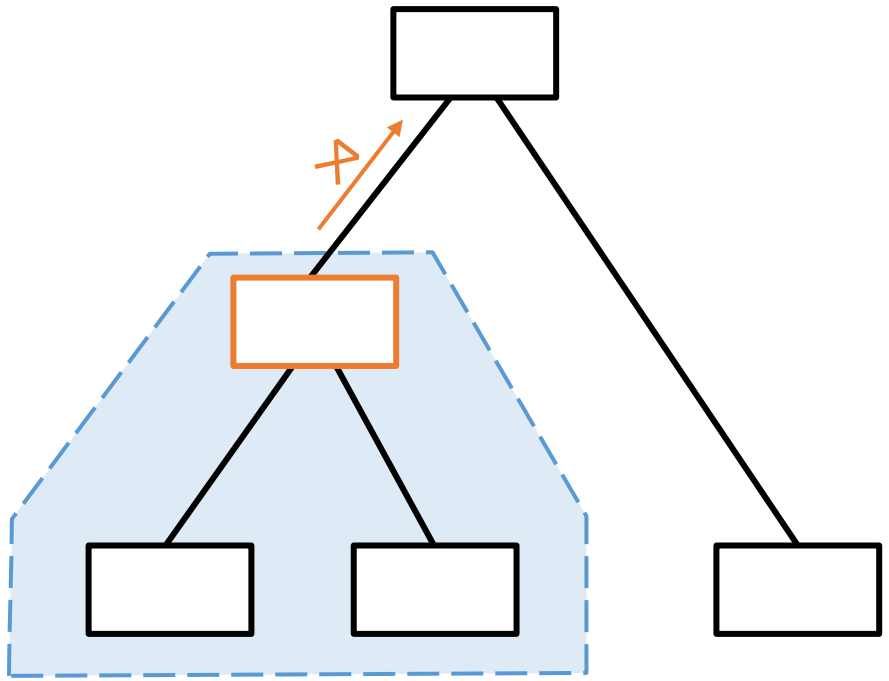
4. down

5. The rewritten query is

$$Q(x,y,z) = R_1(x,y) \bowtie S_2(y,z) \bowtie T_1(z,w)$$

Notice that with 2(k-1) messages, every of the k tables has received information from every other table!

Semijoin Reduction

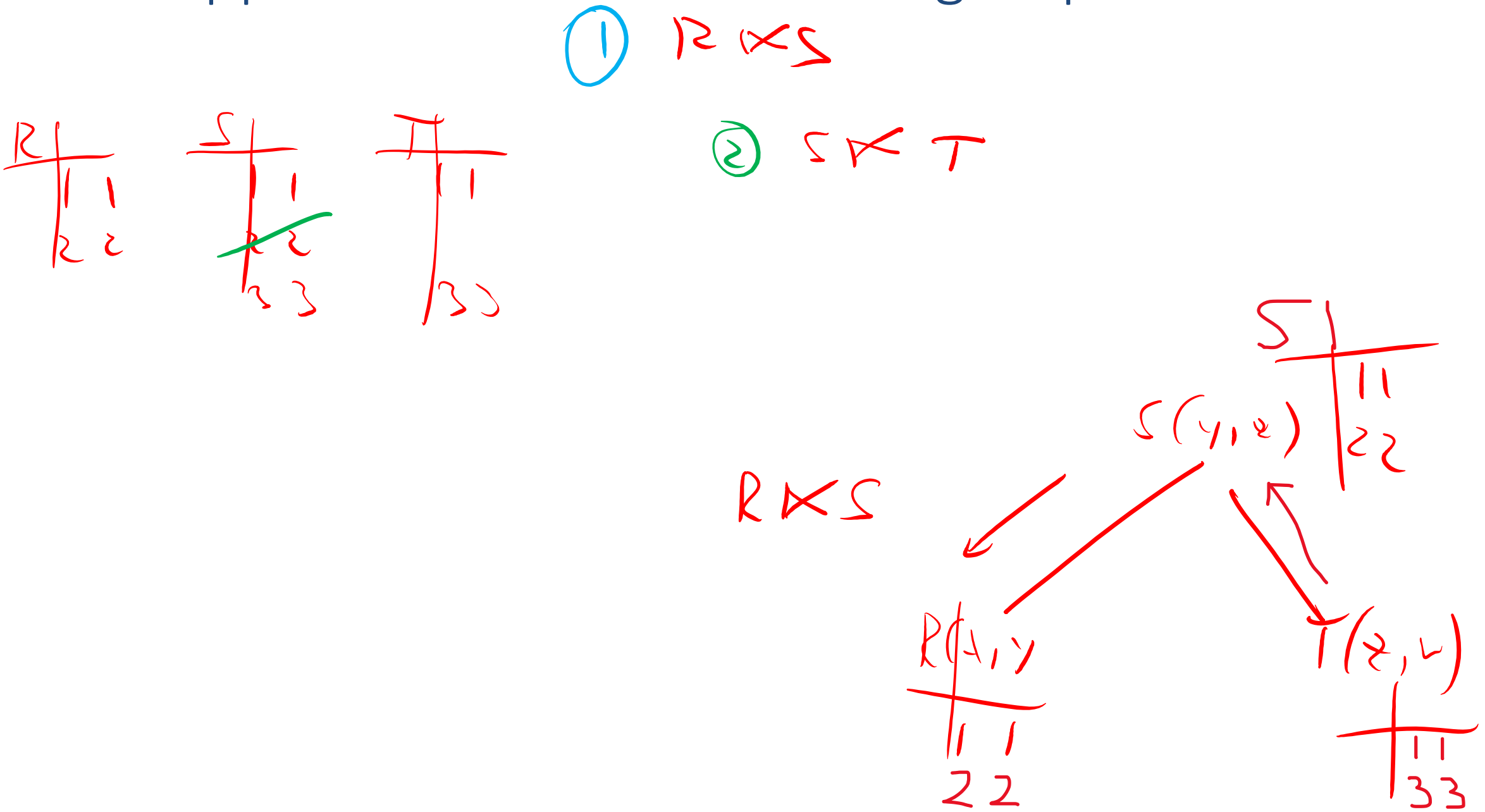


The intuition of the sequence of semi-join reductions is as follows:

- a message is sent across an edge only if the subtree is already fully reduced
- a message that is sent across an edge contains *all the information* from the subtree rooted at the sender.

It follows that the reduction always needs to start at the leaves!

What happens if one uses the wrong sequence?



Semi-join reducers

GYO ear removal

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

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

Join tree

?

Query hypergraph

?

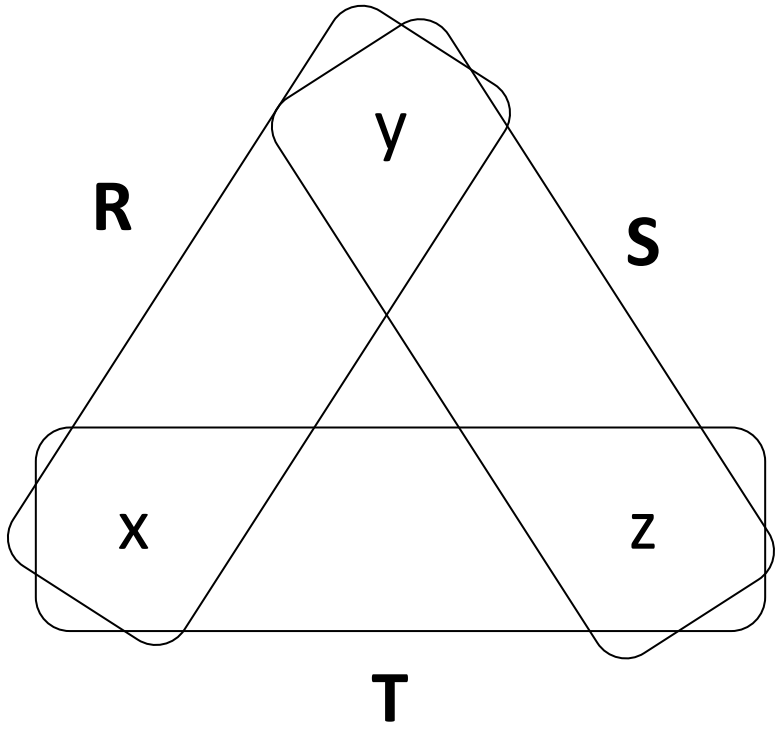
Semi-join reducers

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

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

Join tree

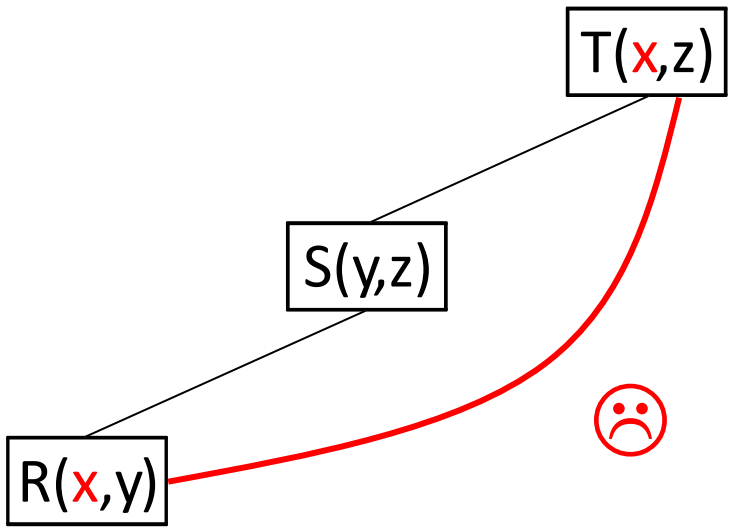
?

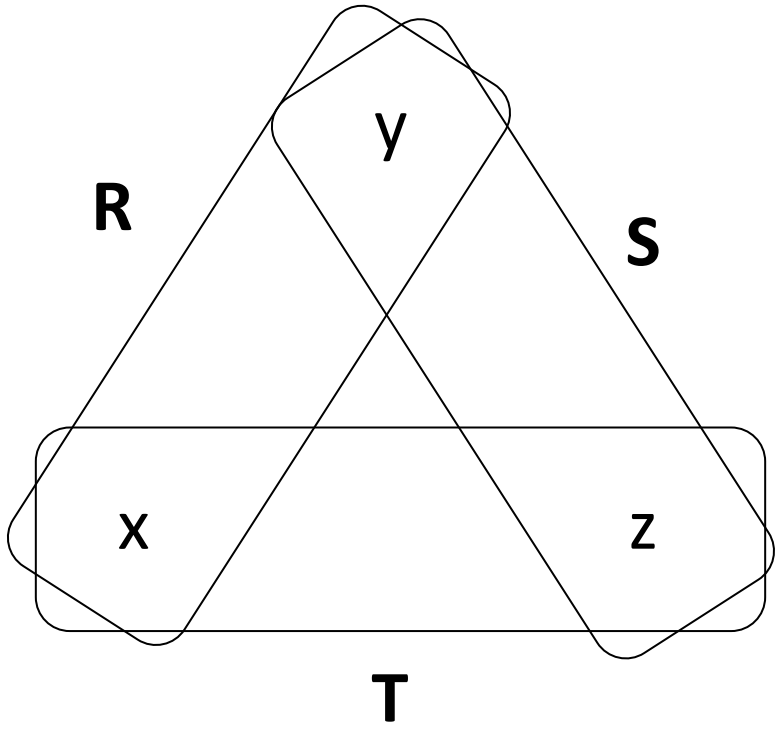


Semi-join reducers

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

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





There is no join tree! You can't fulfill the running intersection property...

Semi-join reducers

GYO ear removal

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

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

Join tree

?

Query hypergraph

?

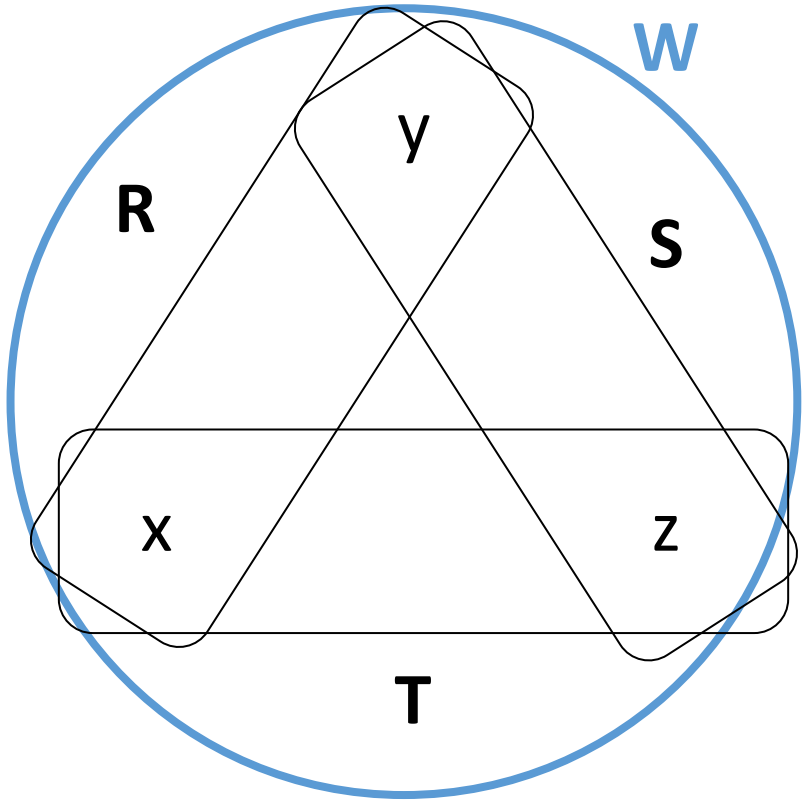
Semi-join reducers

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

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

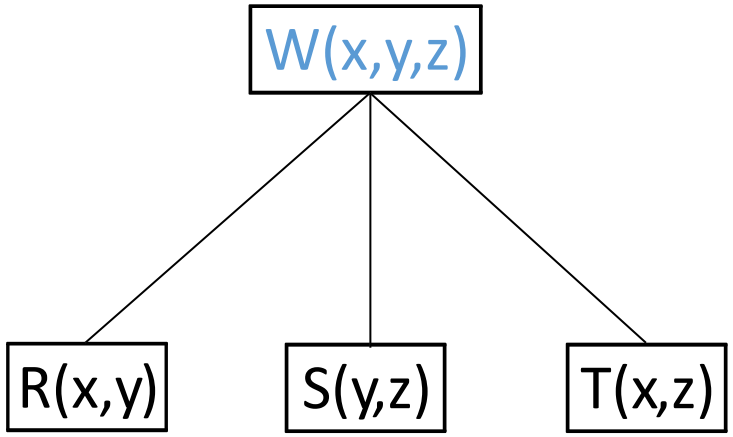
Join tree

?



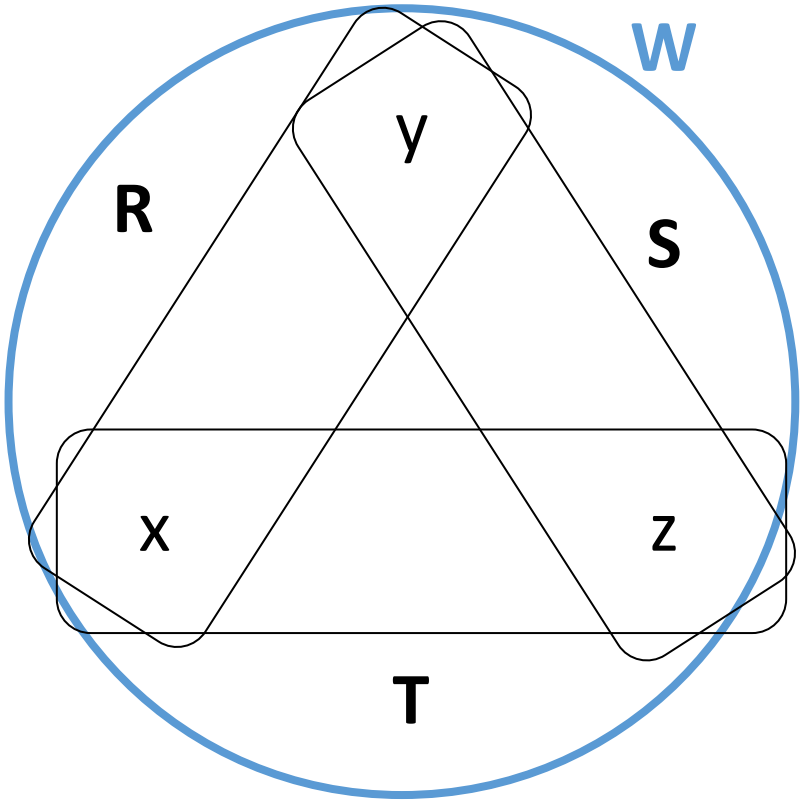
Semi-join reducers

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



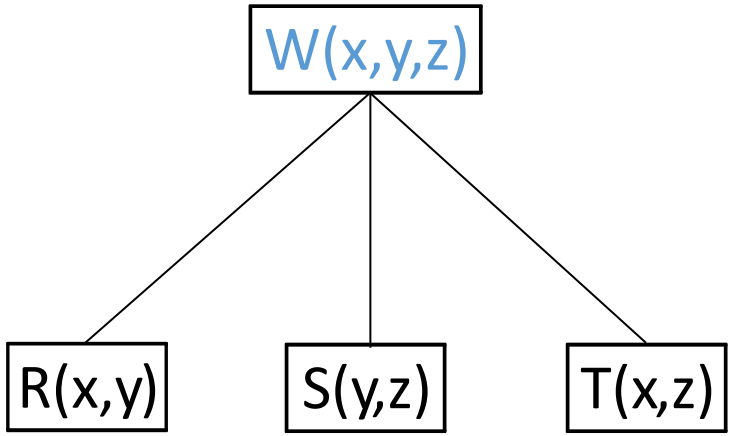
?

Write 1) a full reducer and then 2) the new join expression in RA



Semi-join reducers

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



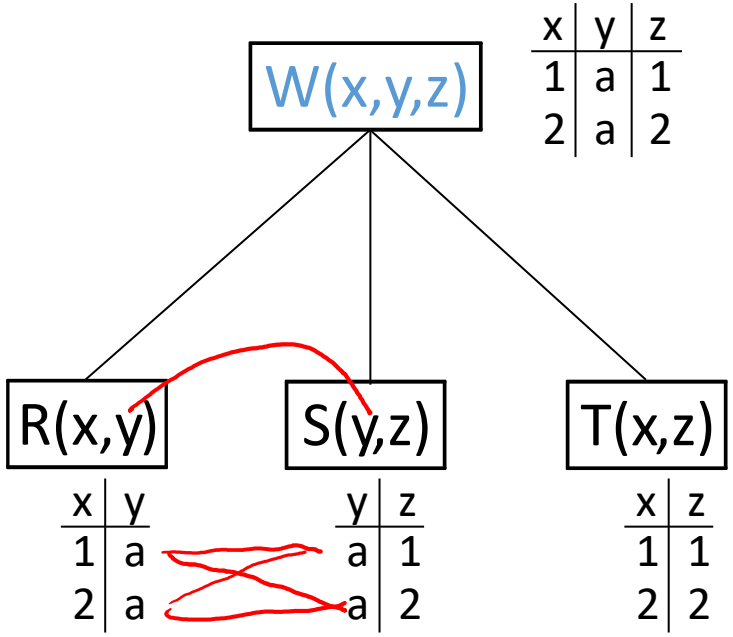
$$W_1(x,y,z) = W(x,y,z) \ltimes R(x,y)$$
$$W_2(x,y,z) = W_1(x,y,z) \ltimes S(y,z)$$
$$W_3(x,y,z) = W_2(x,y,z) \ltimes T(x,z)$$
$$R_1(x,y) = R(x,y) \ltimes W_3(x,y,z)$$
$$S_1(y,z) = S(y,z) \ltimes W_3(x,y,z)$$
$$T_1(x,z) = T(x,z) \ltimes W_3(x,y,z)$$

$$Q(x,y,z) = \Big(\big(R_1(x,y) \bowtie S_1(y,z)\big) \bowtie T_1(x,z)\Big) \bowtie W_3(x,y,z)$$

?

Semi-join reducers

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



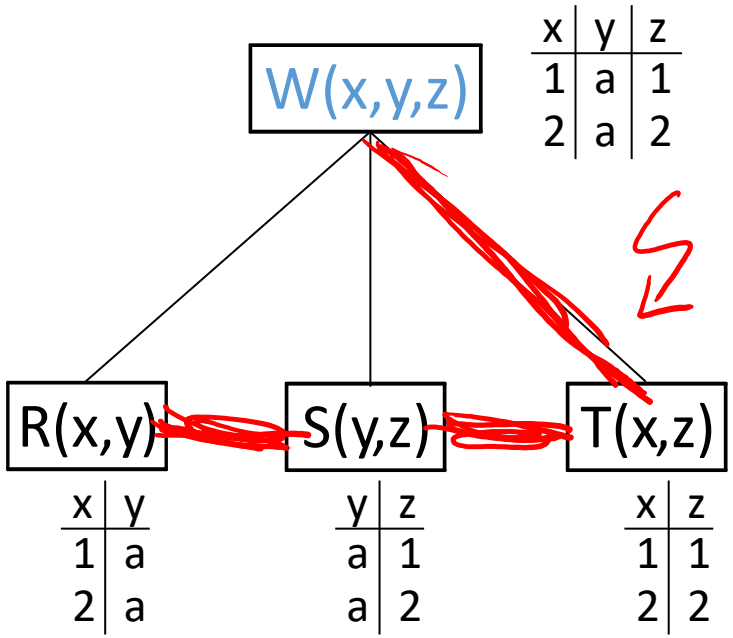
$$W_1(x,y,z) = W(x,y,z) \ltimes R(x,y)$$
$$W_2(x,y,z) = W_1(x,y,z) \ltimes S(y,z)$$
$$W_3(x,y,z) = W_2(x,y,z) \ltimes T(x,z)$$
$$R_1(x,y) = R(x,y) \ltimes W_3(x,y,z)$$
$$S_1(y,z) = S(y,z) \ltimes W_3(x,y,z)$$
$$T_1(x,z) = T(x,z) \ltimes W_3(x,y,z)$$

$$Q(x,y,z) = \left(\left(R_1(x,y) \bowtie S_1(y,z)\right) \bowtie T_1(x,z)\right) \bowtie W_3(x,y,z)$$

?

Semi-join reducers

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



x	y	z
1	a	1
2	a	2

x	y
1	a
2	a

y	z
a	1
a	2

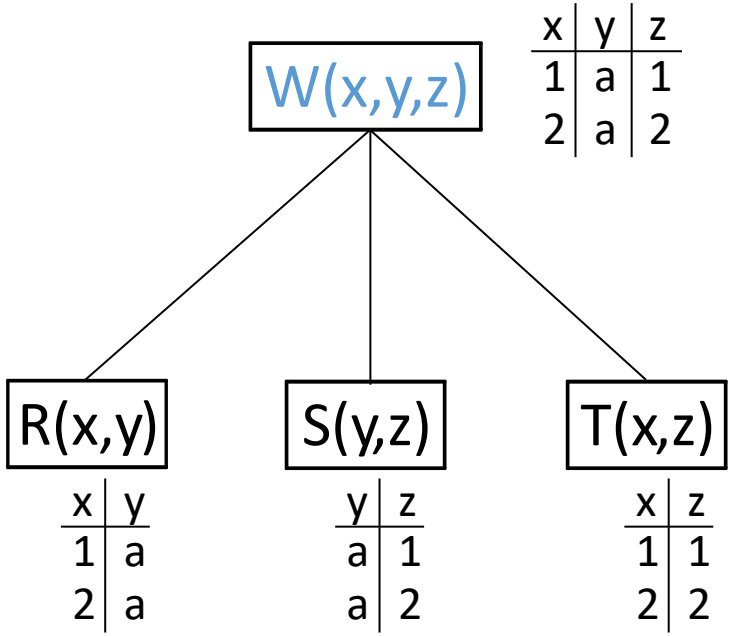
x	z
1	1
2	2

$$W_1(x,y,z) = W(x,y,z) \ltimes R(x,y)$$
$$W_2(x,y,z) = W_1(x,y,z) \ltimes S(y,z)$$
$$W_3(x,y,z) = W_2(x,y,z) \ltimes T(x,z)$$
$$R_1(x,y) = R(x,y) \ltimes W_3(x,y,z)$$
$$S_1(y,z) = S(y,z) \ltimes W_3(x,y,z)$$
$$T_1(x,z) = T(x,z) \ltimes W_3(x,y,z)$$

$$Q(x,y,z) = \left(\left(R_1(x,y) \bowtie S_1(y,z)\right) \bowtie T_1(x,z)\right) \bowtie W_3(x,y,z)$$

Semi-join reducers

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



x	y	z
1	a	1
2	a	2

x	y
1	a
2	a

y	z
a	1
a	2

x	z
1	1
2	2

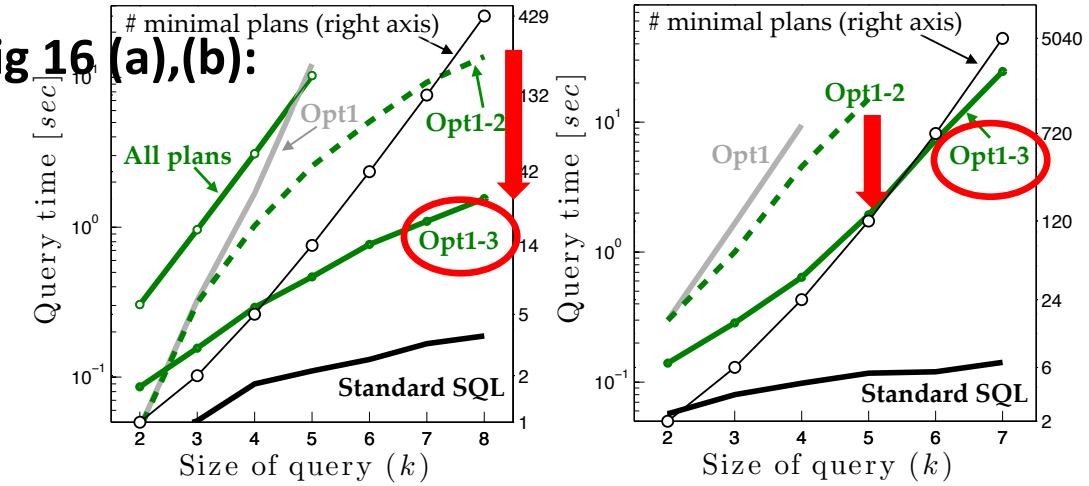
$$W_1(x,y,z) = W(x,y,z) \ltimes R(x,y)$$
$$W_2(x,y,z) = W_1(x,y,z) \ltimes S(y,z)$$
$$W_3(x,y,z) = W_2(x,y,z) \ltimes T(x,z)$$
$$R_1(x,y) = R(x,y) \ltimes W_3(x,y,z)$$
$$S_1(y,z) = S(y,z) \ltimes W_3(x,y,z)$$
$$T_1(x,z) = T(x,z) \ltimes W_3(x,y,z)$$

$$Q(x,y,z) = (R_1(x,y) \bowtie S_1(y,z)) \bowtie T_1(x,z)) \bowtie W_3(x,y,z)$$

$$Q(x,y,z) = (R_1(x,y) \bowtie W_3(x,y,z)) \bowtie S_1(y,z)) \bowtie T_1(x,z)$$

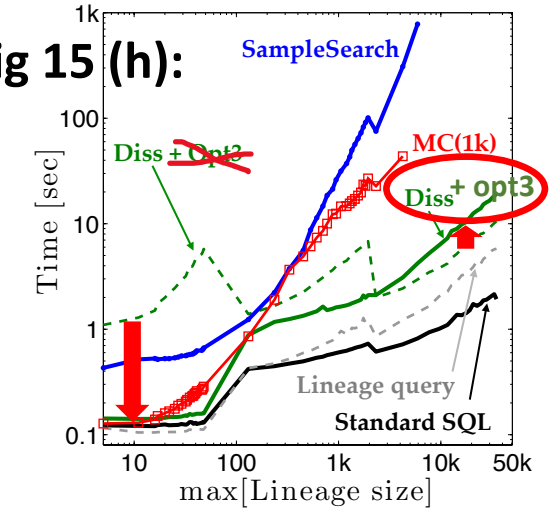
Semi-join reductions can be extremely powerful

Fig 16 (a),(b):



(a) k-chain queries (b) k-star queries

Fig 15 (h):



(h) Combining (a)-(c)

Optimization is speculative (i.e. depends on the inputs to pay off)

6.3 Opt. 3: Deterministic semi-join reduction

The most expensive operations in probabilistic query plans are the group-bys for the probabilistic project operations. These are often applied early in the plans to tuples which are later pruned and do not contribute to the final query result. Our third optimization is to first apply a *full semi-join reduction on the input relations* before starting the probabilistic evaluation from these *reduced input relations*.

We like to draw here an important connection to [54], which introduces the idea of "lazy plans" and shows orders of magnitude performance improvements for safe plans by computing confidences not after each join and projection, but rather at the very end of the plan. We note that our semi-join reduction *serves the same purpose* with similar performance improvements and also apply for safe queries. The advantage of semi-join reductions, however, is that we do not require any modifications to the query engine.

From: Gatterbauer, Suciu. "Dissociation and propagation for approximate lifted inference with standard relational database management systems", VLDBJ 2017. https://arxiv.org/pdf/1310.6257
Reference [54]: Olteanu, Huang, Koch. "Sprout: Lazy vs. eager query plans for tuple-independent probabilistic databases", ICDE 2009. https://doi.org/10.1109/ICDE.2009.123