CS294-248 Special Topics in Database Theory
Unit 2: Conjunctive Queries

Dan Suciu

University of Washington

Query Evaluation for CQ

Motivation

We already know that the data complexity is in AC^0.

What is the expression complexity? The combined complexity?

Will answer both, and also discuss the expression/combined complexity for FO (which we left out).

Importantly: we will define query evaluation for CQ in terms of Homomorphisms

Equivalent Concepts

- A Conjunctive Query:
  $R(x, y, z) \wedge S(x, u) \wedge S(y, v) \wedge S(z, w) \wedge R(u, v, w)$
- A database instance:

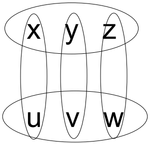
$R(A, B, C) =$

A	B	C
x	y	z
u	v	w

$S(D, E) =$

D	E
x	u
y	v
z	w

- A labeled hypergraph, $G = (V, E)$, where
  $V = \{x, y, z, u, v, w\}$, $E = \{\{x, y, z\}, \{u, v, w\}, \{x, u\}, \{y, v\}, \{z, w\}\}$
  (hyperedges are labeled with R, S respectively).



We will often switch back-and-forth between these equivalent notions

Equivalent Concepts

- A Conjunctive Query:
  $R(x, y, z) \wedge S(x, u) \wedge S(y, v) \wedge S(z, w) \wedge R(u, v, w)$
- A database instance:

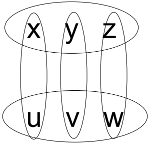
$R(A, B, C) =$

A	B	C
x	y	z
u	v	w

$S(D, E) =$

D	E
x	u
y	v
z	w

- A labeled hypergraph, $G = (V, E)$, where
  $V = \{x, y, z, u, v, w\}$, $E = \{\{x, y, z\}, \{u, v, w\}, \{x, u\}, \{y, v\}, \{z, w\}\}$
  (hyperedges are labeled with R, S respectively).



We will often switch back-and-forth between these equivalent notions

Homomorphisms

$$Q(\boldsymbol{x}_0) = R_1(\boldsymbol{x}_1) \wedge \cdots \wedge R_m(\boldsymbol{x}_m),\ Q'(\boldsymbol{y}_0) = S_1(\boldsymbol{y}_1) \wedge \cdots \wedge S_n(\boldsymbol{y}_n).$$

Definition

A homomorphism $h : Q' \to Q$ is a function
$h : \texttt{Const}(Q') \cup \texttt{Vars}(Q') \to \texttt{Const}(Q) \cup \texttt{Vars}(Q)$ s.t.:

- $\forall c \in \texttt{Const}(Q')$, $h(c) = c$.
- $S_j(\boldsymbol{y}_j) \in \texttt{Atoms}(Q')$, $\exists R_i(\boldsymbol{x}_i) \in \texttt{Atoms}(Q)$ such that $R_i = S_j$ (the are the same relation name) and $h(\boldsymbol{y}_j) = \boldsymbol{x}_i$.
- h maps head vars to head vars: $h(\boldsymbol{y}_0) = \boldsymbol{x}_0$.

Graph homomorphism $h : G' \to G$ is $h : V \to V'$ s.t. $\forall e \in E'$, $h(e) \in E$.

Homomorphisms

$$Q(\boldsymbol{x}_0) = R_1(\boldsymbol{x}_1) \wedge \cdots \wedge R_m(\boldsymbol{x}_m), \; Q'(\boldsymbol{y}_0) = S_1(\boldsymbol{y}_1) \wedge \cdots \wedge S_n(\boldsymbol{y}_n).$$

Definition

A homomorphism $h : Q' \to Q$ is a function
$h : \texttt{Const}(Q') \cup \texttt{Vars}(Q') \to \texttt{Const}(Q) \cup \texttt{Vars}(Q)$ s.t.:

- $\forall c \in \texttt{Const}(Q')$, $h(c) = c$.
- $S_j(\boldsymbol{y}_j) \in \texttt{Atoms}(Q')$, $\exists R_i(\boldsymbol{x}_i) \in \texttt{Atoms}(Q)$ such that $R_i = S_j$ (the are the same relation name) and $h(\boldsymbol{y}_j) = \boldsymbol{x}_i$.
- h maps head vars to head vars: $h(\boldsymbol{y}_0) = \boldsymbol{x}_0$.

Graph homomorphism $h : G' \to G$ is $h : V \to V'$ s.t. $\forall e \in E'$, $h(e) \in E$.

Query Evaluation for CQ and Homomorphisms

Computing $Q(\boldsymbol{D})$ consists of finding all homomorphisms $h : Q \to D$ and returning $h(\texttt{Head}(Q))$.

$Q(x) = R(x) \wedge S(x, y) \wedge T(y, {'a'})$

$R =$

x
1
2

$S =$

x	y
1	10
1	20
2	20

$T =$

y	z
10	a
10	b
20	a

We list all homomorphisms:

$h =$

$x(= \texttt{Head}(Q))$	y	a
1	10	a
1	20	a
2	20	a

Final answer after duplicate elimination: $Q(\boldsymbol{D}) = \{1, 2\}$.

The Combined Complexity for UCQ is in NP

Theorem

The combined complexity for UCQ is in NP.

Proof: Fix a UCQ $Q = Q_1 \vee Q_2 \vee \cdots$ and a database $\boldsymbol{D}$.

To check $\boldsymbol{D} \models Q$:

- "guess" a CQ Q_i, and
- "guess" a homomorphism $h : Q_i \to \boldsymbol{D}$

The Expression Complexity for CQ is NP-hard

Theorem

There exists a database $\boldsymbol{D}$ for which the expression complexity of CQ queries is NP complete.

Thus, the expression complexity is also NP-complete.

Proof Many proofs are possible (will explain shortly why). We will use reduction from 3SAT, because we will reuse it a few times.

Given a 3CNF formula Φ we construct $Q_\Phi, \boldsymbol{D}$ such that:

$$\boxed{\Phi \text{ is satisfiable iff } \exists h : Q_\Phi \to \boldsymbol{D}.}$$

Notice that D is independent of Φ.

Details next.

The Expression Complexity for CQ is NP-hard

Theorem

There exists a database $\boldsymbol{D}$ for which the expression complexity of CQ queries is NP complete.

Thus, the expression complexity is also NP-complete.

Proof Many proofs are possible (will explain shortly why). We will use reduction from 3SAT, because we will reuse it a few times.

Given a 3CNF formula Φ we construct $Q_\Phi, \boldsymbol{D}$ such that:

$$\boxed{\Phi \text{ is satisfiable iff } \exists h : Q_\Phi \to \boldsymbol{D}.}$$

Notice that D is independent of Φ.

Details next.

The Expression Complexity for CQ is NP-hard

Theorem

There exists a database $\boldsymbol{D}$ for which the expression complexity of CQ queries is NP complete.

Thus, the expression complexity is also NP-complete.

Proof Many proofs are possible (will explain shortly why). We will use reduction from 3SAT, because we will reuse it a few times.

Given a 3CNF formula Φ we construct $Q_\Phi, \boldsymbol{D}$ such that:

$$\boxed{\Phi \text{ is satisfiable iff } \exists h : Q_\Phi \rightarrow \boldsymbol{D}.}$$

Notice that D is independent of Φ.

Details next.

Reduction from 3SAT to CQ Evaluation

Given a 3CNF formula Φ we construct $Q_\Phi, \boldsymbol{D}$ such that:

$$\boxed{\Phi \text{ is satisfiable iff } \exists h : Q_\Phi \to \boldsymbol{D}.}$$

Q_Φ has one atom for each clause C in Φ:

- If $C = (X_i \vee X_j \vee X_k)$ then Q_Φ contains $A(x_i, x_j, x_k)$.
- If $C = (X_i \vee X_j \vee \neg X_k)$ then Q_Φ contains $B(x_i, x_j, x_k)$.
- If $C = (X_i \vee \neg X_j \vee \neg X_k)$ then Q_Φ contains $C(x_i, x_j, x_k)$.
- If $C = (\neg X_i \vee \neg X_j \vee \neg X_k)$ then Q_Φ contains $D(x_i, x_j, x_k)$.

$\boldsymbol{D}$ has 4 tables with 7 tuples each which tuple is missing?

$A =$

0	0	1
	$\vdots$	
1	1	1

$B =$

0	0	0
0	1	0
	$\vdots$	
1	1	1

$C = \ldots$ $\qquad$ $D = \ldots$

In class: Φ is satisfiable iff $\exists h : Q \to \boldsymbol{D}$.

Reduction from 3SAT to CQ Evaluation

Given a 3CNF formula Φ we construct $Q_\Phi, \boldsymbol{D}$ such that:

$$\boxed{\Phi \text{ is satisfiable iff } \exists h : Q_\Phi \to \boldsymbol{D}.}$$

Q_Φ has one atom for each clause C in Φ:

- If $C = (X_i \vee X_j \vee X_k)$ then Q_Φ contains $A(x_i, x_j, x_k)$.
- If $C = (X_i \vee X_j \vee \neg X_k)$ then Q_Φ contains $B(x_i, x_j, x_k)$.
- If $C = (X_i \vee \neg X_j \vee \neg X_k)$ then Q_Φ contains $C(x_i, x_j, x_k)$.
- If $C = (\neg X_i \vee \neg X_j \vee \neg X_k)$ then Q_Φ contains $D(x_i, x_j, x_k)$.

$\boldsymbol{D}$ has 4 tables with 7 tuples each which tuple is missing?

$A =$

0	0	1
	⋮	
1	1	1

$B =$

0	0	0
0	1	0
	⋮	
1	1	1

$C = \dots$ $D = \dots$

In class: Φ is satisfiable iff $\exists h : Q \to \boldsymbol{D}$.

Reduction from 3SAT to CQ Evaluation

Given a 3CNF formula Φ we construct $Q_\Phi, \boldsymbol{D}$ such that:

$$\boxed{\Phi \text{ is satisfiable iff } \exists h : Q_\Phi \to \boldsymbol{D}.}$$

Q_Φ has one atom for each clause C in Φ:

- If $C = (X_i \vee X_j \vee X_k)$ then Q_Φ contains $A(x_i, x_j, x_k)$.
- If $C = (X_i \vee X_j \vee \neg X_k)$ then Q_Φ contains $B(x_i, x_j, x_k)$.
- If $C = (X_i \vee \neg X_j \vee \neg X_k)$ then Q_Φ contains $C(x_i, x_j, x_k)$.
- If $C = (\neg X_i \vee \neg X_j \vee \neg X_k)$ then Q_Φ contains $D(x_i, x_j, x_k)$.

$\boldsymbol{D}$ has 4 tables with 7 tuples each which tuple is missing?

$$A = \begin{array}{|c|c|c|} 0 & 0 & 1 \\ & \vdots & \\ 1 & 1 & 1 \end{array} \qquad B = \begin{array}{|c|c|c|} 0 & 0 & 0 \\ 0 & 1 & 0 \\ & \vdots & \\ 1 & 1 & 1 \end{array} \qquad C = \ldots \qquad D = \ldots$$

In class: Φ is satisfiable iff $\exists h : Q \to \boldsymbol{D}$.

Reduction from 3SAT to CQ Evaluation

Given a 3CNF formula Φ we construct $Q_\Phi, \boldsymbol{D}$ such that:

$$\boxed{\Phi \text{ is satisfiable iff } \exists h : Q_\Phi \rightarrow \boldsymbol{D}.}$$

Q_Φ has one atom for each clause C in Φ:

- If $C = (X_i \vee X_j \vee X_k)$ then Q_Φ contains $A(x_i, x_j, x_k)$.
- If $C = (X_i \vee X_j \vee \neg X_k)$ then Q_Φ contains $B(x_i, x_j, x_k)$.
- If $C = (X_i \vee \neg X_j \vee \neg X_k)$ then Q_Φ contains $C(x_i, x_j, x_k)$.
- If $C = (\neg X_i \vee \neg X_j \vee \neg X_k)$ then Q_Φ contains $D(x_i, x_j, x_k)$.

$\boldsymbol{D}$ has 4 tables with 7 tuples each which tuple is missing?

$$A = \begin{array}{|c|c|c|} \hline 0 & 0 & 1 \\ & \vdots & \\ 1 & 1 & 1 \\ \hline \end{array} \qquad B = \begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ 0 & 1 & 0 \\ & \vdots & \\ 1 & 1 & 1 \\ \hline \end{array} \qquad C = \ldots \qquad D = \ldots$$

In class: Φ is satisfiable iff $\exists h : Q \rightarrow \boldsymbol{D}$.

Reduction from 3SAT to CQ Evaluation

Given a 3CNF formula Φ we construct $Q_\Phi, \boldsymbol{D}$ such that:

$$\boxed{\Phi \text{ is satisfiable iff } \exists h : Q_\Phi \to \boldsymbol{D}.}$$

Q_Φ has one atom for each clause C in Φ:

- If $C = (X_i \vee X_j \vee X_k)$ then Q_Φ contains $A(x_i, x_j, x_k)$.
- If $C = (X_i \vee X_j \vee \neg X_k)$ then Q_Φ contains $B(x_i, x_j, x_k)$.
- If $C = (X_i \vee \neg X_j \vee \neg X_k)$ then Q_Φ contains $C(x_i, x_j, x_k)$.
- If $C = (\neg X_i \vee \neg X_j \vee \neg X_k)$ then Q_Φ contains $D(x_i, x_j, x_k)$.

$\boldsymbol{D}$ has 4 tables with 7 tuples each which tuple is missing?

$$A = \begin{array}{|c|c|c|} 0 & 0 & 1 \\ & \vdots & \\ 1 & 1 & 1 \\ \hline \end{array} \qquad B = \begin{array}{|c|c|c|} 0 & 0 & 0 \\ 0 & 1 & 0 \\ & \vdots & \\ 1 & 1 & 1 \\ \hline \end{array} \qquad C = \ldots \qquad D = \ldots$$

In class: Φ is satisfiable iff $\exists h : Q \to \boldsymbol{D}$.

Combined Complexity for FO

Recall that the combined complexity of FO is in PSPACE.

Theorem

There exists a database $\boldsymbol{D}$ for which the expression complexity of FO queries is PSPACE complete.

Thus, the combined complexity is also PSPACE-complete.

Proof: Reduction from the Quantified Boolean Formula Satfiability:

$$\boxed{Q_1 X_1 \; Q_2 X_2 \; \cdots Q_n X_n \; \Phi}$$

where Φ is 3CNF.

Use the same $Q_\Phi, \boldsymbol{D}$ before, but add appropriate quantifiers to Q_Φ:

$$\boxed{Q x_1 \; Q x_2 \; \cdots Q_n x_n \; Q_\Phi(x_1, \ldots, x_n)}$$

Combined Complexity for FO

Recall that the combined complexity of FO is in PSPACE.

Theorem

There exists a database $\boldsymbol{D}$ for which the expression complexity of FO queries is PSPACE complete.

Thus, the combined complexity is also PSPACE-complete.

Proof: Reduction from the Quantified Boolean Formula Satfiability:

$$\boxed{Q_1 X_1 \ Q_2 X_2 \ \cdots Q_n X_n \ \Phi}$$

where Φ is 3CNF.

Use the same $Q_\Phi, \boldsymbol{D}$ before, but add appropriate quantifiers to Q_Φ:

$$\boxed{Q x_1 \ Q x_2 \ \cdots Q_n x_n \ Q_\Phi(x_1, \ldots, x_n)}$$

Combined Complexity for FO

Recall that the combined complexity of FO is in PSPACE.

Theorem

There exists a database $\boldsymbol{D}$ *for which the expression complexity of FO queries is PSPACE complete.*

Thus, the combined complexity is also PSPACE-complete.

Proof: Reduction from the Quantified Boolean Formula Satfiability:

$$\boxed{Q_1 X_1 \ Q_2 X_2 \ \cdots Q_n X_n \ \Phi}$$

where Φ is 3CNF.

Use the same $Q_\Phi, \boldsymbol{D}$ before, but add appropriate quantifiers to Q_Φ:

$$\boxed{Q x_1 \ Q x_2 \ \cdots Q_n x_n \ Q_\Phi(x_1, \ldots, x_n)}$$

Discussion: CQ and CSP

The generalized Constraint Satisfaction Problem is:

Definition ([Kolaitis and Vardi, 1998])

Given two classes of finite structures $\mathcal{A}, \mathcal{B}$, the $CSP(\mathcal{A}, \mathcal{B})$ problem is:
Given $A \in \mathcal{A}, B \in \mathcal{B}$, is there a homomorphism $h : A \to B$?

Standard CSP restricts the right-hand side, $CSP(-, B)$.
What is B for 3SAT? For 3-colorability? For Hamiltonean path?

Query evaluation restricts the left-hand side, $CSP(Q, -)$

"Query evaluation is CSP from the other side."

Discussion: CQ and CSP

The generalized Constraint Satisfaction Problem is:

Definition ([Kolaitis and Vardi, 1998])

Given two classes of finite structures $\mathcal{A}, \mathcal{B}$, the $CSP(\mathcal{A}, \mathcal{B})$ problem is:
Given $A \in \mathcal{A}, B \in \mathcal{B}$, is there a homomorphism $h : A \to B$?

Standard CSP restricts the right-hand side, $CSP(-, B)$.
What is B for 3SAT? For 3-colorability? For Hamiltonean path?

Query evaluation restricts the left-hand side, $CSP(Q, -)$

"Query evaluation is CSP from the other side."

Discussion: CQ and CSP

The generalized Constraint Satisfaction Problem is:

Definition ([Kolaitis and Vardi, 1998])

Given two classes of finite structures $\mathcal{A}, \mathcal{B}$, the $CSP(\mathcal{A}, \mathcal{B})$ problem is:
Given $A \in \mathcal{A}, B \in \mathcal{B}$, is there a homomorphism $h : A \to B$?

Standard CSP restricts the right-hand side, $CSP(-, B)$.
What is B for 3SAT? For 3-colorability? For Hamiltonean path?

Query evaluation restricts the left-hand side, $CSP(Q, -)$

"Query evaluation is CSP from the other side."

Summary

- Evaluating $Q(\boldsymbol{D})$ consists of finding homomorphisms $h : Q \to \boldsymbol{D}$.
- This problem is in NP, in fact it is the very definition of NP.
- If Q is fixed, then the problem is in PTIME in $|\boldsymbol{D}|$. Data complexity
- If Q is part of the input (i.e. can be huge) then NP-complete. Expression complexity

Acyclic Queries

Motivation

How efficiently can we compute a conjunctive query Q on a database $\boldsymbol{D}$?
$N \stackrel{\text{def}}{=} |\text{ADom}(\boldsymbol{D})|$, $M \stackrel{\text{def}}{=} \max_i |R_i^D|$.

- Nested for-loops:

```
for x1 in ADom
    for x2 in ADom
        ...
```

  Runtime: $O\left(N^{|\text{Vars}(Q)|}\right)$.

- Joins: $(\ldots(R_1 \bowtie R_2) \bowtie R_2 \ldots) \bowtie R_m$

  Runtime:[1] $\tilde{O}\left(M^{|\text{Atoms}(Q)|}\right)$.

Both are $O\left(\text{Input}|^{O(1)}\right)$. We would like: $\boxed{\tilde{O}(|\text{Input}| + |\text{Output}|)}$

Semijoin reduction, and tree decomposition.

[1] Recall: $\tilde{O}(f(N))$ means $O(f(N) \log N)$.

Motivation

How efficiently can we compute a conjunctive query Q on a database $\boldsymbol{D}$?
$N \stackrel{\text{def}}{=} |\text{ADom}(\boldsymbol{D})|$, $M \stackrel{\text{def}}{=} \max_i |R_i^D|$.

- Nested for-loops:

```
for x1 in ADom
    for x2 in ADom
        ...
```

  Runtime: $O\left(N^{|\text{Vars}(Q)|}\right)$.

- Joins: $(\ldots(R_1 \bowtie R_2) \bowtie R_2 \ldots) \bowtie R_m$

  Runtime:[1] $\tilde{O}\left(M^{|\text{Atoms}(Q)|}\right)$.

Both are $O\left(|\text{Input}|^{O(1)}\right)$. We would like: $\boxed{\tilde{O}(|\text{Input}| + |\text{Output}|)}$

Semijoin reduction, and tree decomposition.

[1] Recall: $\tilde{O}(f(N))$ means $O(f(N)\log N)$.

Motivation

How efficiently can we compute a conjunctive query Q on a database $\boldsymbol{D}$?
$N \stackrel{\text{def}}{=} |\text{ADom}(\boldsymbol{D})|$, $M \stackrel{\text{def}}{=} \max_i |R_i^D|$.

- Nested for-loops:

```
for x1 in ADom
    for x2 in ADom
        ...
```

  Runtime: $O\left(N^{|\text{Vars}(Q)|}\right)$.

- Joins: $\boxed{(\ldots(R_1 \bowtie R_2) \bowtie R_2 \ldots) \bowtie R_m}$

  Runtime:[1] $\tilde{O}\left(M^{|\text{Atoms}(Q)|}\right)$.

Both are $O\left(\text{Input}|^{O(1)}\right)$. We would like: $\boxed{\tilde{O}(|\text{Input}| + |\text{Output}|)}$

Semijoin reduction, and tree decomposition.

[1] Recall: $\tilde{O}(f(N))$ means $O(f(N) \log N)$.

Motivation

How efficiently can we compute a conjunctive query Q on a database $\boldsymbol{D}$?
$N \stackrel{\text{def}}{=} |\text{ADom}(\boldsymbol{D})|$, $M \stackrel{\text{def}}{=} \max_i |R_i^D|$.

- Nested for-loops:

  Runtime: $O\left(N^{|\text{Vars}(Q)|}\right)$.

```
for x1 in ADom
    for x2 in ADom
        ...
```

- Joins:

  Runtime:[1] $\tilde{O}\left(M^{|\text{Atoms}(Q)|}\right)$.

$$\boxed{(\ldots(R_1 \bowtie R_2) \bowtie R_2 \ldots) \bowtie R_m}$$

Both are $O\left(|\text{Input}|^{O(1)}\right)$. We would like: $\boxed{\tilde{O}(|\text{Input}| + |\text{Output}|)}$

Semijoin reduction, and tree decomposition.

[1]Recall: $\tilde{O}(f(N))$ means $O(f(N) \log N)$.

Motivation

How efficiently can we compute a conjunctive query Q on a database $\boldsymbol{D}$?
$N \stackrel{\text{def}}{=} |\text{ADom}(\boldsymbol{D})|$, $M \stackrel{\text{def}}{=} \max_i |R_i^D|$.

- Nested for-loops:

$$\boxed{\begin{array}{l}\texttt{for } x_1 \text{ in ADom} \\ \quad \texttt{for } x_2 \text{ in ADom} \\ \quad\quad \ldots\end{array}}$$

  Runtime: $O\left(N^{|\text{Vars}(Q)|}\right)$.

- Joins:

$$\boxed{(\ldots(R_1 \bowtie R_2) \bowtie R_2 \ldots) \bowtie R_m}$$

  Runtime:[1] $\tilde{O}\left(M^{|\text{Atoms}(Q)|}\right)$.

Both are $O\left(|\text{Input}|^{O(1)}\right)$. We would like: $\boxed{\tilde{O}(|\text{Input}| + |\text{Output}|)}$

Semijoin reduction, and tree decomposition.

[1] Recall: $\tilde{O}(f(N))$ means $O(f(N) \log N)$.

Joins, Semijoins

Suppose relations $A(\boldsymbol{x}, \boldsymbol{y}), B(\boldsymbol{x}, \boldsymbol{z})$ have common variables $\boldsymbol{x}$.

Definition

Join $A \bowtie B$: $J(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = A(\boldsymbol{x}, \boldsymbol{y}) \wedge B(\boldsymbol{x}, \boldsymbol{z})$.
(Left) Semi-join $SJ = A \ltimes B$: $SJ(\boldsymbol{x}, \boldsymbol{y}) = A(\boldsymbol{x}, \boldsymbol{y}) \wedge B(\boldsymbol{x}, \boldsymbol{z})$.

Fact

$A \bowtie B$ can be computed in time $\tilde{O}(|A| + |B| + |A \bowtie B|)$.
$A \ltimes B$ can be computed in time $\tilde{O}(|A| + |B|)$.

Joins, Semijoins: Properties

- $A \ltimes B \subseteq A$. $\quad A := A \ltimes B$ doesn't increase size.
- $A \bowtie B = (A \ltimes B) \bowtie B$. $\quad A := A \ltimes B$ doesn't affect the join.
- $A \ltimes B = \Pi_{\text{Vars}(A)}(A \bowtie B)$. $\quad A := A \ltimes B$ is reduced for $A \bowtie B$.

Joins, Semijoins: Properties

- $A \ltimes B \subseteq A$. $A := A \ltimes B$ doesn't increase size.
- $A \bowtie B = (A \ltimes B) \bowtie B$. $A := A \ltimes B$ doesn't affect the join.
- $A \ltimes B = \Pi_{\text{Vars}(A)}(A \bowtie B)$. $A := A \ltimes B$ is reduced for $A \bowtie B$.
- Idempotence: $(A \ltimes B) \ltimes B = A \ltimes B$

Joins, Semijoins: Properties

- $A \ltimes B \subseteq A$. $A := A \ltimes B$ doesn't increase size.
- $A \bowtie B = (A \ltimes B) \bowtie B$. $A := A \ltimes B$ doesn't affect the join.
- $A \ltimes B = \Pi_{\text{Vars}(A)}(A \bowtie B)$. $A := A \ltimes B$ is reduced for $A \bowtie B$.
- Idempotence: $(A \ltimes B) \ltimes B = A \ltimes B$
- Does cascading hold? $\boxed{A \ltimes (B \bowtie C) = (A \ltimes B) \ltimes C}$

Joins, Semijoins: Properties

- $A \ltimes B \subseteq A$. $A := A \ltimes B$ doesn't increase size.
- $A \bowtie B = (A \ltimes B) \bowtie B$. $A := A \ltimes B$ doesn't affect the join.
- $A \ltimes B = \Pi_{\mathrm{Vars}(A)}(A \bowtie B)$. $A := A \ltimes B$ is reduced for $A \bowtie B$.
- Idempotence: $(A \ltimes B) \ltimes B = A \ltimes B$
- Does cascading hold? $\boxed{A \ltimes (B \bowtie C) = (A \ltimes B) \ltimes C}$

Yes, when $\mathrm{Vars}(A) \cap \mathrm{Vars}(C) \subseteq \mathrm{Vars}(B)$.

Joins, Semijoins: Properties

- $A \ltimes B \subseteq A$. $A := A \ltimes B$ doesn't increase size.
- $A \bowtie B = (A \ltimes B) \bowtie B$. $A := A \ltimes B$ doesn't affect the join.
- $A \ltimes B = \Pi_{\text{Vars}(A)}(A \bowtie B)$. $A := A \ltimes B$ is reduced for $A \bowtie B$.
- Idempotence: $(A \ltimes B) \ltimes B = A \ltimes B$
- Does cascading hold? $\boxed{A \ltimes (B \bowtie C) = (A \ltimes B) \ltimes C}$

  Yes, when $\text{Vars}(A) \cap \text{Vars}(C) \subseteq \text{Vars}(B)$.
- If $\text{Vars}(A) = \text{Vars}(B)$, what is $A \ltimes B$?

Joins, Semijoins: Properties

- $A \ltimes B \subseteq A$. $A := A \ltimes B$ doesn't increase size.
- $A \bowtie B = (A \ltimes B) \bowtie B$. $A := A \ltimes B$ doesn't affect the join.
- $A \ltimes B = \Pi_{\text{Vars}(A)}(A \bowtie B)$. $A := A \ltimes B$ is reduced for $A \bowtie B$.
- Idempotence: $(A \ltimes B) \ltimes B = A \ltimes B$
- Does cascading hold? $\boxed{A \ltimes (B \bowtie C) = (A \ltimes B) \ltimes C}$

  Yes, when $\text{Vars}(A) \cap \text{Vars}(C) \subseteq \text{Vars}(B)$.
- If $\text{Vars}(A) = \text{Vars}(B)$, what is $A \ltimes B$? $A \ltimes B = B \ltimes A = A \cap B$.

Joins, Semijoins: Properties

- $A \ltimes B \subseteq A$. $A := A \ltimes B$ doesn't increase size.
- $A \bowtie B = (A \ltimes B) \bowtie B$. $A := A \ltimes B$ doesn't affect the join.
- $A \ltimes B = \Pi_{\text{Vars}(A)}(A \bowtie B)$. $A := A \ltimes B$ is reduced for $A \bowtie B$.
- Idempotence: $(A \ltimes B) \ltimes B = A \ltimes B$
- Does cascading hold? $\boxed{A \ltimes (B \bowtie C) = (A \ltimes B) \ltimes C}$

  Yes, when $\text{Vars}(A) \cap \text{Vars}(C) \subseteq \text{Vars}(B)$.
- If $\text{Vars}(A) = \text{Vars}(B)$, what is $A \ltimes B$? $A \ltimes B = B \ltimes A = A \cap B$.
- Does distributivity hold? $\boxed{A \ltimes (B \bowtie C) = (A \ltimes B) \cap (A \ltimes C)}$

Joins, Semijoins: Properties

- $A \ltimes B \subseteq A$. $\quad A := A \ltimes B$ doesn't increase size.
- $A \bowtie B = (A \ltimes B) \bowtie B$. $\quad A := A \ltimes B$ doesn't affect the join.
- $A \ltimes B = \Pi_{\text{Vars}(A)}(A \bowtie B)$. $\quad A := A \ltimes B$ is reduced for $A \bowtie B$.
- Idempotence: $(A \ltimes B) \ltimes B = A \ltimes B$
- Does cascading hold? $\boxed{A \ltimes (B \bowtie C) = (A \ltimes B) \ltimes C}$

  Yes, when $\text{Vars}(A) \cap \text{Vars}(C) \subseteq \text{Vars}(B)$.
- If $\text{Vars}(A) = \text{Vars}(B)$, what is $A \ltimes B$? $\quad A \ltimes B = B \ltimes A = A \cap B$.
- Does distributivity hold? $\boxed{A \ltimes (B \bowtie C) = (A \ltimes B) \cap (A \ltimes C)}$

  Yes, when $\text{Vars}(B) \cap \text{Vars}(C) \subseteq \text{Vars}(A)$.

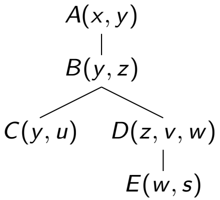
Acyclic Query

Definition

Q is acyclic if it admits a join tree, which is a tree T where:

- The nodes in T are in 1-1 correspondence with the atoms in Q.
- T satisfies the running intersection property: for any variable, the set of nodes that contain it forms a connected component.

Acyclic: $Q = A(x, y) \wedge B(y, z) \wedge C(y, u) \wedge D(z, v, w) \wedge E(w, s)$



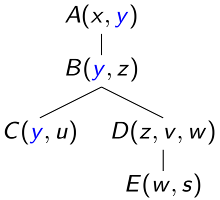
Acyclic Query

Definition

Q is acyclic if it admits a join tree, which is a tree T where:

- The nodes in T are in 1-1 correspondence with the atoms in Q.
- T satisfies the running intersection property: for any variable, the set of nodes that contain it forms a connected component.

Acyclic: $Q = A(x,y) \wedge B(y,z) \wedge C(y,u)$
$\wedge D(z,v,w) \wedge E(w,s)$



E.g. running intersection for y

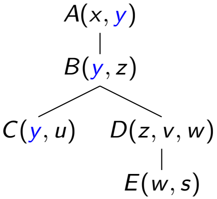
Acyclic Query

Definition

Q is acyclic if it admits a join tree, which is a tree T where:

- The nodes in T are in 1-1 correspondence with the atoms in Q.
- T satisfies the running intersection property: for any variable, the set of nodes that contain it forms a connected component.

Acyclic: $Q = A(x, y) \wedge B(y, z) \wedge C(y, u) \wedge D(z, v, w) \wedge E(w, s)$



E.g. running intersection for y

Not acyclic: $A(x, y) \wedge B(y, z) \wedge C(z, x)$. why?

Acyclic Query - GYO

GYO Acyclicity Test (Graham and Yu-Oszoyoglu)

Repeat:

- Remove an isolated variable (i.e. occurs in only one atom).
- Remove an ear (i.e. atom contain in another atom).

Q is a acyclic iff result is one empty edge. **Proof: exercise.**

Which var is isolated? $Q = A(x, y) \wedge B(y, z) \wedge C(y, u) \wedge D(z, v, w) \wedge E(w, s)$

Acyclic Query - GYO

GYO Acyclicity Test (Graham and Yu-Oszoyoglu)

Repeat:

- Remove an isolated variable (i.e. occurs in only one atom).
- Remove an ear (i.e. atom contain in another atom).

Q is a acyclic iff result is one empty edge. **Proof: exercise.**

Which var is isolated? $Q = A(x, y) \wedge B(y, z) \wedge C(y, u) \wedge D(z, v, w) \wedge E(w, s)$

Acyclic Query - GYO

GYO Acyclicity Test (Graham and Yu-Oszoyoglu)

Repeat:

- Remove an isolated variable (i.e. occurs in only one atom).
- Remove an ear (i.e. atom contain in another atom).

Q is a acyclic iff result is one empty edge. **Proof: exercise.**

$$Q = A(x, y) \wedge B(y, z) \wedge C(y, u) \wedge D(z, v, w) \wedge E(w, s)$$

Which atom is an ear? $\rightarrow A(y) \wedge B(y, z) \wedge C(y, u) \wedge D(z, v, w) \wedge E(w, s)$

Acyclic Query - GYO

GYO Acyclicity Test (Graham and Yu-Oszoyoglu)

Repeat:

- Remove an isolated variable (i.e. occurs in only one atom).
- Remove an ear (i.e. atom contain in another atom).

Q is a acyclic iff result is one empty edge. **Proof: exercise.**

$$Q = A(x, y) \wedge B(y, z) \wedge C(y, u) \wedge D(z, v, w) \wedge E(w, s)$$

Which atom is an ear? $\rightarrow A(y) \wedge B(y, z) \wedge C(y, u) \wedge D(z, v, w) \wedge E(w, s)$

Acyclic Query - GYO

GYO Acyclicity Test (Graham and Yu-Oszoyoglu)

Repeat:

- Remove an isolated variable (i.e. occurs in only one atom).
- Remove an ear (i.e. atom contain in another atom).

Q is a acyclic iff result is one empty edge. **Proof: exercise.**

$$
\begin{aligned}
Q = & A(x, y) \wedge B(y, z) \wedge C(y, u) \wedge D(z, v, w) \wedge E(w, s) \\
\rightarrow & A(y) \wedge B(y, z) \wedge C(y, u) \wedge D(z, v, w) \wedge E(w, s) \\
\rightarrow & B(y, z) \wedge C(y, u) \wedge D(z, v, w) \wedge E(w, s) \\
\rightarrow & B(y, z) \wedge C(y) \wedge D(z, w) \wedge E(w) \\
\rightarrow & B(y, z) \wedge D(z, w) \\
\rightarrow & B(z) \wedge D(z) \\
\rightarrow & D(z) \\
\rightarrow & - \quad \text{Acyclic!}
\end{aligned}
$$

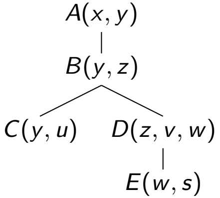
Yannakakis' Algorithm: Boolean Query

Boolean, acyclic query $Q() = \exists x_1 \exists x_2 \cdots$, join tree T.
How do we compute $Q(D)$ in time $O(\texttt{Input})$?

Yannakakis' Algorithm: Boolean Query

Boolean, acyclic query $Q() = \exists x_1 \exists x_2 \cdots$, join tree T.
How do we compute $Q(D)$ in time $O(\texttt{Input})$?

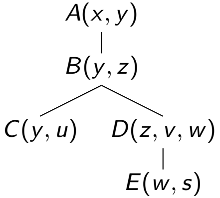
Bottom-up Semi-join Reduction:



Yannakakis' Algorithm: Boolean Query

Boolean, acyclic query $Q() = \exists x_1 \exists x_2 \cdots$, join tree T.
How do we compute $Q(D)$ in time $O(\texttt{Input})$?

Bottom-up Semi-join Reduction:

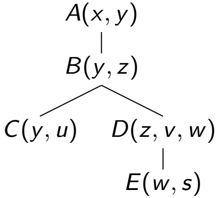


$$D := D \ltimes E$$
$$B := B \ltimes C$$
$$B := B \ltimes D$$
$$A := A \ltimes B$$

Yannakakis' Algorithm: Boolean Query

Boolean, acyclic query $Q() = \exists x_1 \exists x_2 \cdots$, join tree T.
How do we compute $Q(D)$ in time $O(\texttt{Input})$?

Bottom-up Semi-join Reduction:



$$D := D \ltimes E$$
$$B := B \ltimes C$$
$$B := B \ltimes D$$
$$A := A \ltimes B$$

Correctness:

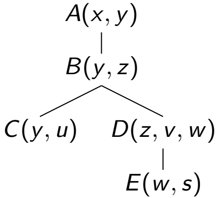
- $A \bowtie (\cdots) \neq \emptyset$ iff $A \ltimes (\cdots) \neq \emptyset$.
- $A \ltimes (B \bowtie (\cdots)) = A \ltimes (B \ltimes (\cdots))$ running intersection property.
- Etc.

Yannakakis' Algorithm: Full Conjunctive Query

Full CQ Q, join tree T, database $\boldsymbol{D}$.

Want to compute $Q(\boldsymbol{D})$ in time $O(|\texttt{Input}| + |\texttt{Output}|$.

Can we simply compute all the joins, in some order?

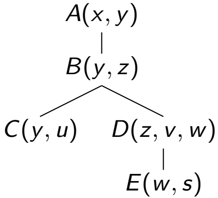


Yannakakis' Algorithm: Full Conjunctive Query

Full CQ Q, join tree T, database $\boldsymbol{D}$.

Want to compute $Q(\boldsymbol{D})$ in time $O(|\texttt{Input}| + |\texttt{Output}|$.

Can we simply compute all the joins, in some order?



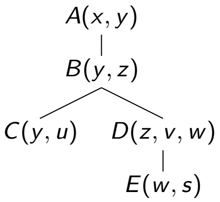
$$
\begin{aligned}
\text{Out}_0 &:= \{()\} \\
\text{Out}_1 &:= \text{Out}_0 \bowtie A \\
\text{Out}_2 &:= \text{Out}_1 \bowtie B \\
\text{Out}_3 &:= \text{Out}_2 \bowtie C \\
\text{Out}_4 &:= \text{Out}_3 \bowtie D \\
Q &:= \text{Out}_4 \bowtie E
\end{aligned}
$$

Yannakakis' Algorithm: Full Conjunctive Query

Full CQ Q, join tree T, database $\boldsymbol{D}$.

Want to compute $Q(\boldsymbol{D})$ in time $O(|\texttt{Input}| + |\texttt{Output}|$.

Can we simply compute all the joins, in some order?



$$
\begin{aligned}
\mathrm{Out}_0 &:= \{()\} \\
\mathrm{Out}_1 &:= \mathrm{Out}_0 \bowtie A \\
\mathrm{Out}_2 &:= \mathrm{Out}_1 \bowtie B \\
\mathrm{Out}_3 &:= \mathrm{Out}_2 \bowtie C \\
\mathrm{Out}_4 &:= \mathrm{Out}_3 \bowtie D \\
Q &:= \mathrm{Out}_4 \bowtie E
\end{aligned}
$$

NO: intermediate results $\gg |\texttt{Output}|$.

Yannakakis' Algorithm: Full Conjunctive Query

Full CQ Q, join tree T, database $\boldsymbol{D}$. Choose an arbitrary root in T.

Phase 1: Semijoin Reduction.

- Traverse the tree bottom-up and set $R_n := R_n \ltimes R_{\texttt{child}(n)}$.
- Traverse the tree top-down and set $R_n := R_n \ltimes R_{\texttt{parent}(n)}$.

Yannakakis' Algorithm: Full Conjunctive Query

Full CQ Q, join tree T, database $\boldsymbol{D}$. Choose an arbitrary root in T.

Phase 1: Semijoin Reduction.

- Traverse the tree bottom-up and set $R_n := R_n \ltimes R_{\texttt{child}(n)}$.
- Traverse the tree top-down and set $R_n := R_n \ltimes R_{\texttt{parent}(n)}$.

Phase 2: Join Computation. Initialize $\text{Out}_0 := \{()\}$ (empty tuple).

- Traverse the tree top-down and set $\text{Out}_i := \text{Out}_{i-1} \bowtie R_n$.

Return Out_m.

Yannakakis' Algorithm: Full Conjunctive Query

Full CQ Q, join tree T, database $\boldsymbol{D}$. Choose an arbitrary root in T.

Phase 1: Semijoin Reduction.

- Traverse the tree bottom-up and set $R_n := R_n \ltimes R_{\texttt{child}(n)}$.
- Traverse the tree top-down and set $R_n := R_n \ltimes R_{\texttt{parent}(n)}$.

Phase 2: Join Computation. Initialize $\text{Out}_0 := \{()\}$ (empty tuple).

- Traverse the tree top-down and set $\text{Out}_i := \text{Out}_{i-1} \bowtie R_n$.

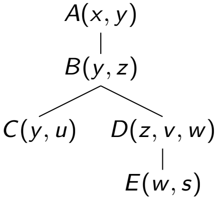
Return Out_m.

Theorem

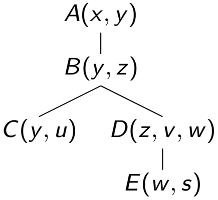
Yannakakis' algorithm is correct and runs in time $O(|\mathtt{Input}| + |\mathtt{Output}|)$

Before the proof, let's see an example.

Yannakakis' Algorithm: Example



Yannakakis' Algorithm: Example



Semijoin Reduction
Bottom-up:

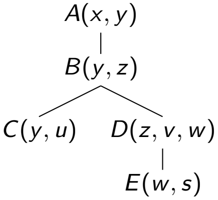
$$D := D \ltimes E$$

$$B := B \ltimes C$$

$$B := B \ltimes D$$

$$A := A \ltimes B$$

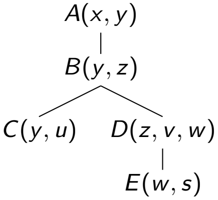
Yannakakis' Algorithm: Example



Semijoin Reduction

Bottom-up:	Top-down:
$D := D \ltimes E$	$B := B \ltimes A$
$B := B \ltimes C$	$C := C \ltimes B$
$B := B \ltimes D$	$D := D \ltimes B$
$A := A \ltimes B$	$E := E \ltimes D$

Yannakakis' Algorithm: Example



Semijoin Reduction

Bottom-up:

$$D := D \ltimes E$$
$$B := B \ltimes C$$
$$B := B \ltimes D$$
$$A := A \ltimes B$$

Top-down:

$$B := B \ltimes A$$
$$C := C \ltimes B$$
$$D := D \ltimes B$$
$$E := E \ltimes D$$

Join Computation

$$\mathrm{Out}_0 := \{()\}$$
$$\mathrm{Out}_1 := \mathrm{Out}_0 \bowtie A$$
$$\mathrm{Out}_2 := \mathrm{Out}_1 \bowtie B$$
$$\mathrm{Out}_3 := \mathrm{Out}_2 \bowtie C$$
$$\mathrm{Out}_4 := \mathrm{Out}_3 \bowtie D$$
$$Q := \mathrm{Out}_4 \bowtie E$$

Yannakakis' Algorithm: Proof

Many proofs are done using informal arguments.

But database optimizers do not understand informal arguments: they are based on identities, or rewrite rules.

Yannakakis' algorithm uses Joins and Semijoins, and we know what identities they satisfy.

Let's prove the correctness and runtime of the algorithm using only those identities.

Yannakakis' Algorithm: Proof

Theorem

Yannakakis' algorithm is correct and runs in time $O(|\mathtt{Input}| + |\mathtt{Output}|)$

Correctness

- If we run only Phase 2, then correctness by assoc./commutativity:

  E.g. $(((D \bowtie A) \bowtie C) \bowtie E) \bowtie B = (((A \bowtie B) \bowtie C) \bowtie D) \bowtie E$

- Phase 1 harmless because $R_i := R_i \ltimes R_j$ does not affect the join.

  E.g. $(((A \bowtie B) \bowtie C) \bowtie D) \bowtie E = (((A \bowtie B) \bowtie (C \ltimes B)) \bowtie D) \bowtie E$

This proves correctness.

Yannakakis' Algorithm: Proof

Theorem

Yannakakis' algorithm is correct and runs in time $O(|\mathtt{Input}| + |\mathtt{Output}|)$

Runtime

Call R reduced w.r.t. Q if $R = R \ltimes Q$. The runtime follows from:

- **CLaim 1** After Phase 1, every R_n is reduced w.r.t. the output Q.
- **Claim 2** During Phase 2, every $\mathtt{Out}_i$ is reduced w.r.t. the output Q.

Runtime of Phase 1 is $O(|\text{Input}|)$.

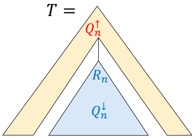
Runtime of Phase 2 is $O(|\text{Input}| + \sum_i |\text{Out}_i|) = O(|\text{Input}| + |\text{Output}|)$.

Proof of Claim 1

For $n \in \text{Nodes}(T)$ define:

$$Q_n^{\downarrow} \stackrel{\text{def}}{=} \Join_{i \in \text{descendants}(n)} R_i$$

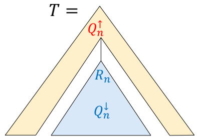
$$Q_n^{\uparrow} \stackrel{\text{def}}{=} \Join_{i \notin \text{descendants}(n)} R_i$$



Proof of Claim 1

For $n \in \text{Nodes}(T)$ define:

$$Q_n^{\downarrow} \stackrel{\text{def}}{=} \Join_{i \in \text{descendants}(n)} R_i$$
$$Q_n^{\uparrow} \stackrel{\text{def}}{=} \Join_{i \notin \text{descendants}(n)} R_i$$



We prove on the next slide:

- After Bottom-up: $\forall n,\ R_n = R_n \ltimes Q_n^{\downarrow}$
- After Top-down: $\forall n,\ R_n = R_n \ltimes Q_n^{\uparrow}$

Therefore, after Phase 1, by distributivity:

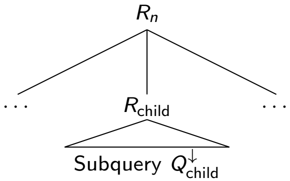
$$R_n \ltimes Q = R_n \ltimes \left(Q_n^{\downarrow} \Join Q_n^{\uparrow}\right) = \left(R_n \ltimes Q_n^{\downarrow}\right) \cap \left(R_n \ltimes Q_n^{\uparrow}\right) = R_n \cap R_n = R_n$$

Details

After Bottom-up, R_n is reduced w.r.t. $Q_n^{\downarrow}$: $R_n = R_n \ltimes Q_n^{\downarrow}$

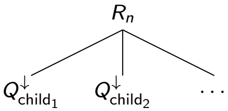
If R_{child} reduced for $Q_{\text{child}}^{\downarrow}$, then so is $R_n^{\text{new}} := R_n \ltimes R_{\text{child}}$:

$$
\begin{aligned}
R_n^{\text{new}} \ltimes Q_{\text{child}}^{\downarrow} &= (R_n \ltimes R_{\text{child}}) \ltimes Q_{\text{child}}^{\downarrow} \\
&= \Big(R_n \ltimes (R_{\text{child}} \ltimes Q_{\text{child}}^{\downarrow})\Big) \ltimes Q_{\text{child}}^{\downarrow} \quad \text{induction} \\
&= \Big(R_n \ltimes (R_{\text{child}} \bowtie Q_{\text{child}}^{\downarrow})\Big) \ltimes Q_{\text{child}}^{\downarrow} \quad \text{cascading} \\
&= \Big(R_n \ltimes (R_{\text{child}} \bowtie Q_{\text{child}}^{\downarrow})\Big) \ltimes (R_{\text{child}} \bowtie Q_{\text{child}}^{\downarrow}) \\
&= R_n \ltimes (R_{\text{child}} \bowtie Q_{\text{child}}^{\downarrow}) = R_n \ltimes R_{\text{child}} = R_n^{\text{new}}
\end{aligned}
$$



If R_n is reduced for each $Q_{\text{child}_i}^{\downarrow}$ then is reduced for $\bowtie_i Q_{\text{child}_i}^{\downarrow}$

$$
\begin{aligned}
R_n \ltimes \Big(\bowtie_i Q_{\text{child}_i}^{\downarrow}\Big) &= \bigcap_i (R_n \ltimes Q_{\text{child}_i}^{\downarrow}) \quad \text{Distributivity} \\
&= R_n
\end{aligned}
$$



After Top-down, R_n is reduced w.r.t. $Q_n^{\uparrow}$: $R_n = R_n \ltimes Q_n^{\uparrow}$. Exercise.

Proof of Claim 2

During Phase 2, Out_i is reduced w.r.t. Q: $\text{Out}_i = \text{Out}_i \ltimes Q$.

By induction on i:

Assuming:

- Induction hypothesis: $\boxed{\text{Out}_i = \text{Out}_i \ltimes Q}$
- By Claim 1: $\boxed{R_n = R_n \ltimes Q}$

prove that $\text{Out}_{i+1} := \text{Out}_i \bowtie R_n$ is reduced w.r.t. Q. Need to show:

$$\boxed{\text{Out}_i \bowtie R_n = (\text{Out}_i \bowtie R_n) \ltimes Q}$$

Does the following hold in general? $(A \bowtie B) \ltimes Q = (A \ltimes Q) \bowtie (B \ltimes Q)$?

Proof of Claim 2

During Phase 2, Out_i is reduced w.r.t. Q: $\text{Out}_i = \text{Out}_i \ltimes Q$.

By induction on i:

Assuming:

- Induction hypothesis: $\boxed{\text{Out}_i = \text{Out}_i \ltimes Q}$
- By Claim 1: $\boxed{R_n = R_n \ltimes Q}$

prove that $\text{Out}_{i+1} := \text{Out}_i \bowtie R_n$ is reduced w.r.t. Q. Need to show:

$$\boxed{\text{Out}_i \bowtie R_n = (\text{Out}_i \bowtie R_n) \ltimes Q}$$

Does the following hold in general? $(A \bowtie B) \ltimes Q = (A \ltimes Q) \bowtie (B \ltimes Q)$?

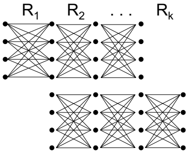
NO!

On Homework 2: complete the proof of Claim 2.

Discussion: is the Semi-join Reduction Necessary?

Yes! Otherwise, intermediate results can be much larger than final result:

$$\text{E.g. } Q(x_0, x_1, \ldots, x_k) = R_1(x_0, x_1) \wedge \cdots \wedge R_k(x_{k-1}, x_k)$$



$$|R_0 \bowtie \cdots \bowtie R_{k-1}| = \Omega(N^k)$$
$$|R_1 \bowtie \cdots \bowtie R_k| = \Omega(N^k)$$
$$R_0 \bowtie R_1 \bowtie \cdots \bowtie R_k = \emptyset$$

$|\text{Input}| = O(N^2)$, $|\text{Output}| = 0$.
If we join directly, then the runtime is $O(N^k) \neq O(|\text{Input}| + |\text{Output}|)$.

Yannakakis Algorithm for General CQ

$$Q(x_1, \ldots, x_p) = \exists x_{p+1} \cdots \exists x_k (A_1 \wedge \cdots \wedge A_m)$$

Definition

Q is acyclic free-connex if it is acyclic after we add an atom $\text{Out}(x_1, \ldots, x_p)$.

Theorem

Yannakis' algorithm computes Q in time $O(|Input| + |Output|)$.

Phase 1 is unchanged. In Phase 2 the elimination order is towards the new atom $\text{Out}(x_1, \ldots, x_p)$.

Example of a Free-Connex Query

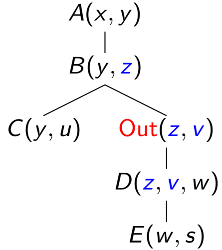
$Q(z, v) =$

$$A(x, y)$$
$$B(y, z)$$
$$C(y, u) \quad D(z, v, w)$$
$$E(w, s)$$

Where do we place
Out(z, v)?

Example of a Free-Connex Query

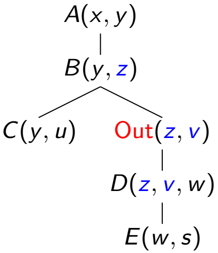
$Q(z, v) =$



Where do we place
Out(z, v)?

Example of a Free-Connex Query

$Q(z, v) =$



Semijoin Reduction

As before.

Example of a Free-Connex Query

$Q(z, v) =$

$A(x, y)$

$B(y, z)$

$C(y, u)$ $\quad$ $\text{Out}(z, v)$

$D(z, v, w)$

$E(w, s)$

Join Computation

$$
\begin{aligned}
T_1(y) &:= A(x, y)\\
T_2(y, z) &:= T_1(y) \bowtie B(y, z)\\
T_3(y) &:= C(y, u)\\
T_4(z) &:= T_2(y, z) \bowtie T_3(y)\\
T_5(w) &:= E(w, s)\\
T_6(z, v) &:= T_5(w) \bowtie D(z, v, w)\\
T_7(z, v) &:= T_6(z, v) \bowtie T_4(z)
\end{aligned}
$$

Return $T_7(z, v)$.

Semijoin Reduction

As before.

The tree traversal is from the leaves towards $\text{Out}(z, v)$. Each T_i is either a subset of some input relation, or of the output $Q(z, v)$, hence Time$= O(|\text{Input}| + |\text{Output}|)$

Non Free-Connex Acylic Queries

If Q is acyclic but not free-connex, unlikely to be computable in time $O(|\text{Input}| + |\text{Output}|)$

Conjecture

The Boolean matrix multiplication conjecture: if A, B are $N \times N$ Boolean matrices, then there exists no algorithm for computing $A \cdot B$ in times $O(N^2)$.

$$\boxed{Q(i,k) = \exists j(A(i,j) \wedge B(j,k))}$$

Cannot compute in time $O(|A| + |B| + |\text{Output}|) = O(N^2)$.

Summary

- Yannakakis' algorithm: Semijoin reduction (up, then down), then joins. For Boolean queries: only upwards semijoin reduction.
- Yannakakis' algorithm is related to the Junction-tree Algorithm in graphical models.
- Most SQL queries in practice are acyclic.
- Discussion in class Do database engines run Yannakakis algorithm? If not, why not?

Hypertree Decomposition

Motivation

What do we do when the query is not acyclic? $R(x, y) \wedge S(y, z) \wedge T(z, x)$.

We compute a tree decomposition then (1) we compute each node of the tree, (2) run Yannakakis' algorithm on the results.

Hypertree Decomposition

Definition

A hypertree decomposition of a query (hypergraph) Q is (T, χ) where T is a tree and $\chi : \text{Nodes}(T) \rightarrow 2^{\text{Vars}(Q)}$ such that:

- Running intersection property: $\forall x \in \text{Vars}(Q)$, the set $\{n \in \text{Nodes}(T) \mid x \in \chi(n)\}$ is connected.
- Every atom $R_i(\boldsymbol{x}_i)$ is covered: $\exists n \in \text{Nodes}(T)$ s.t. $\boldsymbol{x}_i \subseteq \chi(n)$

A set $\chi(n)$ for $n \in \text{Nodes}(T)$ is called a bag.

$$Q = R(x, y) \wedge S(y, z) \wedge T(z, u) \wedge K(u, x)$$

$$T = \quad \begin{array}{c} \{x, y, z\} \\ | \\ \{x, u, z\} \end{array}$$

Hypertree Width

A edge-cover of a set of variables $\boldsymbol{z} \subseteq \text{Vars}(Q)$ is a set $\mathcal{C} \subseteq \text{Atoms}(Q)$ such that $\boldsymbol{z} \subseteq \bigcup_{R(\boldsymbol{x}) \in \mathcal{C}} \boldsymbol{x}$.

The edge-cover number of $\boldsymbol{z}$ is $\rho(\boldsymbol{z}) \stackrel{\text{def}}{=} \min_{\mathcal{C}} |\mathcal{C}|$ where $\mathcal{C}$ ranges over all edge-covers.

Definition

The hypertree width of a tree is $(htw)(T) \stackrel{\text{def}}{=} \max_{n \in \text{Nodes}(T)} \rho(\chi(n))$.

The hypertree width of a query is $(htw)(Q) \stackrel{\text{def}}{=} \min_T (htw)(T)$ where T ranges over tree decompositions of Q.

Warning: some text use the term *generalized* hypertree width.

What is $(htw)(Q)$?

$Q = R(x, y) \wedge S(y, z) \wedge T(z, u) \wedge K(u, x)$

$$\begin{array}{c}\{x, y, z\} \\ | \\ \{x, u, z\}\end{array}$$

Discussion: Structural Optimization of Conjunctive Queries

Assume Q is a full conjunctive query:

- Find a tree decomposition with minimum $(htw)(T)$.
- Compute every bag using a left-deep join plan $(R_1 \bowtie R_2) \bowtie \cdots$ and materialize it.
  (We will discuss a better method, Worst-Case Optimal Joins, in a few weeks. Don't miss it!)
- Run Yannakakis' algorithm on the result.

Query Containment, Equivalence, Minimization

Motivation

Query equivalence means $Q_1(\boldsymbol{D}) = Q_2(\boldsymbol{D})$ for any input database $\boldsymbol{D}$.

This is the most important static analysis problem.

Will show that equivalence is undecidable for FO,
but is decidable for CQ, UCQ, and extensions with inequalities $(\leq, \neq)$.

Query Equivalence

Definition (Equivalence)

Q_1, Q_2 are equivalent if $\forall \boldsymbol{D}$, $Q_1(\boldsymbol{D}) = Q_2(\boldsymbol{D})$. Notation: $Q_1 \equiv Q_2$.

It suffices to study equivalence of Boolean queries, because of the following:

Fact

$Q_1(\boldsymbol{x}) \equiv Q_2(\boldsymbol{y})$ iff they have the same arity ($|\boldsymbol{x}| = |\boldsymbol{y}|$), and for some constants $\boldsymbol{c}$ not occurring in Q_1, Q_2, $Q_1[\boldsymbol{c}/\boldsymbol{x}] \equiv Q_2[\boldsymbol{c}/\boldsymbol{y}]$.

Query Containment

Definition (Containment)

Q_1 is contained in Q_2 if $\forall \boldsymbol{D}$, $Q_1(\boldsymbol{D}) \subseteq Q_2(\boldsymbol{D})$. Notation: $Q_1 \Rightarrow Q_2$

It suffices to assume Q_1, Q_2 are Boolean. Then $Q_1 \subseteq Q_2$ same as $Q_1 \Rightarrow Q_2$.

Fact

Equivalence and containment are (almost) the same problem:

$$\boxed{Q_1 \equiv Q_2} \text{ iff } \boxed{Q_1 \Rightarrow Q_2 \text{ and } Q_2 \Rightarrow Q_1}$$

$$\boxed{Q_1 \Rightarrow Q_2} \text{ iff}^2 \boxed{Q_1 \equiv Q_1 \wedge Q_2}$$

[2] Language must be closed under $\wedge$.

Containment for FO is Undecidable

Theorem

The problem Given Q_1, Q_2, check whether $Q_1 \subseteq Q_2$ is undecidable.

Proof By reduction from $\texttt{SAT}_{\text{fin}}$.

Let Φ be any sentence. (We want to check $\texttt{SAT}_{\text{fin}}(\Phi)$.)

Define $Q_1 \stackrel{\text{def}}{=} \Phi$ and $Q_2 \stackrel{\text{def}}{=} \texttt{false}$. Then $Q_1 \subseteq Q_2$ iff $\texttt{SAT}_{\text{fin}}(\Phi)$.

Containment for CQs

The containment problem for CQ is decidable; More precisely, NP-complete.

This is one of the oldest, most celebrated result in database theory [Chandra and Merlin, 1977].

Containment for CQs

Assume CQs Boolean queries; extension to non-Boolean is immediate.

Definition (Canonical Database)

The canonical database associated to a CQ Q is the following: its domain is $\texttt{Vars}(Q)$, and its tuples are the atoms of Q. Notation: $\boldsymbol{D}_Q$.

Theorem

The following are equivalent:

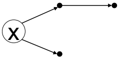
- *Containment holds:* $Q_1 \subseteq Q_2$
- *There exists a homomorphism* $h : Q_2 \to Q_1$
- $Q_2(\boldsymbol{D}_{Q_1}) = \mathtt{true}$.

Proof in class.

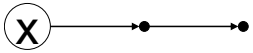
Examples

Which pairs of queries are contained? Equivalent?

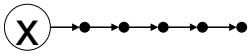
$Q_1(x) = \exists y \exists z \exists w (E(x, y) \wedge E(y, z) \wedge E(x, w))$



$Q_2(x) = \exists u \exists v (E(x, u) \wedge E(u, v))$



$Q_3(x) = \exists u_1 \cdots \exists u_5 (E(x, u_1) \wedge E(u_1, u_2) \wedge \cdots \wedge E(u_4, u_5))$

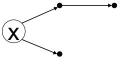


$Q_4(x) = \exists y (E(x, y) \wedge E(y, x))$

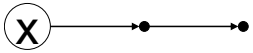
Examples

Which pairs of queries are contained? Equivalent?

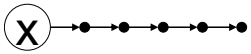
$Q_1(x) = \exists y \exists z \exists w (E(x,y) \wedge E(y,z) \wedge E(x,w))$



$Q_2(x) = \exists u \exists v (E(x,u) \wedge E(u,v))$



$Q_3(x) = \exists u_1 \cdots \exists u_5 (E(x,u_1) \wedge E(u_1,u_2) \wedge \cdots \wedge E(u_4,u_5))$



$Q_4(x) = \exists y (E(x,y) \wedge E(y,x))$

$$Q_4 \subseteq Q_3 \subsetneq Q_1 \equiv Q_2$$

Containment of UCQs

Theorem

If $Q = Q_1 \vee Q_2 \vee \cdots$, $Q' = Q'_1 \vee Q'_2 \vee$ *then the following are equivalent:*

- *Containment holds:* $Q \subseteq Q'$
- *Every* Q_i *is contained in some* Q_j*:* $\forall i \exists j$, $Q_i \subseteq Q'_j$.

Proof in class.

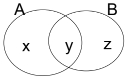
Example: Proving Join/Semi-join Identities

$$\boxed{A \bowtie B = (A \ltimes B) \bowtie B}$$

Example: Proving Join/Semi-join Identities

$$\boxed{A \bowtie B = (A \ltimes B) \bowtie B}$$

Denote $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ the set of variables:



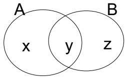
$$\begin{aligned} Q_1(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) =& A(\boldsymbol{x}, \boldsymbol{y}) \wedge B(\boldsymbol{y}, \boldsymbol{z}) \\ Q_2(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) =& (\exists \boldsymbol{z}\, (A(\boldsymbol{x}, \boldsymbol{y}) \wedge B(\boldsymbol{y}, \boldsymbol{z}))) \wedge B(\boldsymbol{y}, \boldsymbol{z}) \\ =& \exists u A(\boldsymbol{x}, \boldsymbol{y}) \wedge B(\boldsymbol{y}, \boldsymbol{u}) \wedge B(\boldsymbol{y}, \boldsymbol{z}) \end{aligned}$$

We renamed $\exists z$ to $\exists u$ so it doesn't clash with the head variable $\boldsymbol{z}$.

Example: Proving Join/Semi-join Identities

$$\boxed{A \bowtie B = (A \ltimes B) \bowtie B}$$

Denote $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ the set of variables:



$$\begin{aligned}
Q_1(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) =& A(\boldsymbol{x}, \boldsymbol{y}) \wedge B(\boldsymbol{y}, \boldsymbol{z}) \\
Q_2(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) =& (\exists \boldsymbol{z}\, (A(\boldsymbol{x}, \boldsymbol{y}) \wedge B(\boldsymbol{y}, \boldsymbol{z}))) \wedge B(\boldsymbol{y}, \boldsymbol{z}) \\
=& \exists u A(\boldsymbol{x}, \boldsymbol{y}) \wedge B(\boldsymbol{y}, \boldsymbol{u}) \wedge B(\boldsymbol{y}, \boldsymbol{z})
\end{aligned}$$

We renamed $\exists z$ to $\exists u$ so it doesn't clash with the head variable $\boldsymbol{z}$.

$h_1 : Q_1 \to Q_2$ maps $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) \mapsto (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$.

$h_2 : Q_2 \to Q_1$ maps $(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{y}, \boldsymbol{z}) \mapsto (\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y}, \boldsymbol{z})$.

Therefore, $\boxed{Q_1 \equiv Q_2}$.

Query Minimization

A CQ Q may be equivalent to many other CQs $Q \equiv Q_2 \equiv Q_3 \equiv \cdots$.

Definition (Minimal Query)

A CQ Q is minimal if $Q \equiv Q'$ implies $|\texttt{Atoms}(Q)| \leq |\texttt{Atoms}(Q')|$.
The minimization problem is: given Q, find $Q_{\min} \equiv Q$ s.t. $Q_{\min}$ is minimal.

E.g. minimize: $Q(x) = \exists y \exists z \exists w (E(x, y) \wedge E(y, z) \wedge E(x, w))$

Query Minimization

A CQ Q may be equivalent to many other CQs $Q \equiv Q_2 \equiv Q_3 \equiv \cdots$.

Definition (Minimal Query)

A CQ Q is minimal if $Q \equiv Q'$ implies $|\texttt{Atoms}(Q)| \leq |\texttt{Atoms}(Q')|$.
The minimization problem is: given Q, find $Q_{\text{min}} \equiv Q$ s.t. Q_{min} is minimal.

E.g. minimize: $Q(x) = \exists y \exists z \exists w (E(x, y) \wedge E(y, z) \wedge E(x, w))$

$Q_{\text{min}} = \exists y \exists z \exists w (E(x, y) \wedge E(y, z))$

Query Minimization

A CQ Q may be equivalent to many other CQs $Q \equiv Q_2 \equiv Q_3 \equiv \cdots$.

Definition (Minimal Query)

A CQ Q is minimal if $Q \equiv Q'$ implies $|\texttt{Atoms}(Q)| \leq |\texttt{Atoms}(Q')|$.
The minimization problem is: given Q, find $Q_{\min} \equiv Q$ s.t. $Q_{\min}$ is minimal.

E.g. minimize: $Q(x) = \exists y \exists z \exists w (E(x,y) \wedge E(y,z) \wedge E(x,w))$

$Q_{\min} = \exists y \exists z \exists w (E(x,y) \wedge E(y,z))$

Theorem

The minimal query is unique up to isomorphism.

Proof: Let Q, Q' minimal and $Q \equiv Q'$; then $\exists h : Q \to Q'$, $h' : Q' \to Q$.
$h' \circ h : Q \to Q$ is surjective, otherwise $Q \equiv \text{Im}(h' \circ h)$ violating minimality.
Thus, $h' \circ h$ is an isomorphism (since its domain is finite).

The Core of a CQ

Definition

The core of Q is a subquery Q_0 (meaning: a subset of atoms) such that
(1) there exists a homomorphism $h : Q \to Q_0$, and
(2) there is no strict subquery of Q_0 with this property.

Note: the term core is commonly used for graphs.

The Core of a CQ

Definition

The core of Q is a subquery Q_0 (meaning: a subset of atoms) such that
(1) there exists a homomorphism $h : Q \to Q_0$, and
(2) there is no strict subquery of Q_0 with this property.

Note: the term core is commonly used for graphs.

Theorem

The core of Q is a minimal query equivalent to Q.

Minimization Algorithm: Repeatedly remove an atom A from Q as long as $\exists h : Q \to Q - \{A\}$.

Minimizing UCQ

A UCQ query $\boxed{Q = Q_1 \vee Q_2 \vee \cdots}$ is minimal if:

- each CQ Q_i is minimal
- for all i, j, $Q_i \subseteq Q_j$ implies $i = j$.

(Discussion in class)

Minimizing UCQ

A UCQ query $\boxed{Q = Q_1 \vee Q_2 \vee \cdots}$ is minimal if:

- each CQ Q_i is minimal
- for all i, j, $Q_i \subseteq Q_j$ implies $i = j$.

(Discussion in class)

Query minimization:

- Minimize each Q_i for $i = 1, 2, \ldots$
- Remove Q_i whenever $\exists j \neq i$ s.t. $Q_i \subseteq Q_j$.

Summary

- Query containment/minimization is the poster child of database theory.

- In practice? Not so much. Real queries have bag semantics query minimization does not apply: $Q_1(x) = R(x) \wedge R(x)$ is not equivalent to $Q_2(x) = R(x)$.

- However the theory becomes quite relevant for reasoning about semi-joins and query rewriting using views, which is a major topic for database systems.

- Next: adding inequalities $\leq, \neq$. The query containment/minimization problem becomes surprisingly subtle!

Adding Inequalities: $<, \leq, \neq$

Inequalities

Extend CQ with $<, \leq, \neq$. E.g. $Q(x, y, z) = R(x, y) \wedge R(x, z) \wedge y \neq z$.

The extend languages is denoted $\text{CQ}^{<}$, or $\text{CQ}^{\leq,\neq}$, or $\text{CQ}(\leq, \neq)$.

The domain of a database instance $\boldsymbol{D}$ is densely ordered, e.g. a subset of $\mathbb{Q}$.

Problems: containment, minimization.

Homomorphism is Sufficient

A homomorphism $h : Q' \to Q$ is now required to map an inequality $t_1 \operatorname{op} t_2$ in Q' to one implied by Q, i.e. $Q \models h(t_1) \operatorname{op} h(t_2)$.

Fact

If there exists a homomorphism $Q' \to Q$ then $Q \subseteq Q'$.

Proof by example. Q, Q' are Boolean queries (dropping $\exists$):

$$Q = R(x, y, z) \wedge x < y \wedge y < z$$

$$Q' = R(u, v, w) \wedge u \leq w$$

The homomorphism $(u, v, w) \mapsto (x, y, z)$ maps $u \leq w$ to $x \leq z$.
We have $Q \models x \leq z$, therefore, $Q \subseteq Q'$

Homomorphism is Not Necessary

Fact

A homomorphism $Q' \to Q$ is a sufficient, but not a necessary condition for $Q \subseteq Q'$.

Homomorphism is Not Necessary

Fact

A homomorphism $Q' \to Q$ is a sufficient, but not a necessary condition for $Q \subseteq Q'$.

Example: (Boolean queries):

$$Q = S(x, y) \wedge S(y, z) \wedge x < z$$
$$Q' = S(u, v) \wedge u < v$$

There is no homomorphism $Q' \to Q$, yet $Q \subseteq Q'$. Why?

Preorder Relations

A relation $\preceq$ on a set V is called a preorder if:

- It is reflexive: $x \preceq x$.
- It is transitive: $x \preceq y$, $y \preceq z$ implies $x \preceq z$.

Write $\boxed{a \equiv b}$ for $a \preceq b$ and $b \preceq a$.
The preorder is total if $\forall a, b \in V$, either $a \preceq b$ or $b \preceq a$ or both hold.

Preorder Relations

A relation $\preceq$ on a set V is called a preorder if:

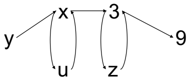
- It is reflexive: $x \preceq x$.
- It is transitive: $x \preceq y$, $y \preceq z$ implies $x \preceq z$.

Write $\boxed{a \equiv b}$ for $a \preceq b$ and $b \preceq a$.
The preorder is total if $\forall a, b \in V$, either $a \preceq b$ or $b \preceq a$ or both hold.

For a preorder $\preceq$ on $\text{Vars}(Q) \cup \text{Const}(Q)$, $Q_{\preceq} \stackrel{\text{def}}{=}$ is its extension with $\preceq$.

E.g. $\boxed{Q = R(x, y, 3) \wedge S(y, z, u, 9) \wedge u \leq x}$

Total preorder: $y \prec x \equiv u \prec 3 \equiv z \prec 9$



$$\boxed{Q_{\preceq} = R(x, y, 3) \wedge S(y, z, u, 9) \wedge y < x \wedge x = u \wedge x < 3 \wedge 3 = z \wedge \cdots}$$

A Necessary and Sufficient Condition

Theorem ([Klug, 1988])

Let Q, Q' be $CQ^{<,\leq,\neq}$ queries. The following conditions are equivalent:

- $Q \subseteq Q'$
- *For any consistent total preorder $\preceq$ on Q, $\exists h : Q' \rightarrow Q_{\preceq}$.*

A Necessary and Sufficient Condition

Theorem ([Klug, 1988])

Let Q, Q' be $CQ^{<,\leq,\neq}$ queries. The following conditions are equivalent:

- $Q \subseteq Q'$
- *For any consistent total preorder $\preceq$ on Q, $\exists h : Q' \rightarrow Q_{\preceq}$.*

Proof: If $Q(\boldsymbol{D}) = \texttt{true}$, then there exists a homomorphism:

$$h_0 : Q \rightarrow \boldsymbol{D}$$

This induces a total preorder $\preceq$ on Q. Let h be a homomorphism:

$$h : Q' \rightarrow Q_{\preceq}$$

Their composition is a homomorphism $Q' \rightarrow \boldsymbol{D}$, proving $Q'(\boldsymbol{D}) = \texttt{true}$.

Example

$$\boxed{Q = S(x, y) \wedge S(y, z) \wedge x < z} \qquad \boxed{Q' = S(u, v) \wedge u < v}$$

Lets prove that $\boxed{Q \subseteq Q'}$.

Example

$$\boxed{Q = S(x,y) \wedge S(y,z) \wedge x < z} \qquad \boxed{Q' = S(u,v) \wedge u < v}$$

Lets prove that $\boxed{Q \subseteq Q'}$.

3 consistent total preorders on Q:

$$\begin{aligned}
Q_1 =& S(x,y) \wedge S(y,z) \wedge x = y \wedge y < z \\
Q_2 =& S(x,y) \wedge S(y,z) \wedge x < y \wedge y < z \\
Q_3 =& S(x,y) \wedge S(y,z) \wedge x < y \wedge y = z
\end{aligned}$$

Example

$\boxed{Q = S(x, y) \wedge S(y, z) \wedge x < z}$ $\boxed{Q' = S(u, v) \wedge u < v}$

Lets prove that $\boxed{Q \subseteq Q'}$.

3 consistent total preorders on Q:

$$\begin{aligned} Q_1 =& S(x, y) \wedge S(y, z) \wedge x = y \wedge y < z \\ Q_2 =& S(x, y) \wedge S(y, z) \wedge x < y \wedge y < z \\ Q_3 =& S(x, y) \wedge S(y, z) \wedge x < y \wedge y = z \end{aligned}$$

In each case, either $(u, v) \mapsto (x, y)$ or $(u, v) \mapsto (y, z)$ is a homomorphism.

Notice: we need to check both homomorphisms.

Complexity

Theorem ([Klug, 1988, van der Meyden, 1997])

The problem given Q, Q' in $CQ^{<,\leq,\neq}$ determine whether $Q \subseteq Q'$ is Π_2^p-complete.

Proof: Membership in Π_2^p follows from the fact that $Q \subseteq Q'$ if for all refinements of Q, there exists a homomorphisms $Q' \to Q$.

For hardness we will discuss a simpler proof than [van der Meyden, 1997].

Proof of Π_2^p-Hardness

Reduction from $\forall 3CNF$: $\boxed{\Psi = \forall X_1 \cdots \forall X_k \exists X_{k+1} \cdots \exists X_n \Phi}$, Φ is 3CNF.

Proof of Π_2^p-Hardness

Reduction from $\forall 3CNF$: $\boxed{\Psi = \forall X_1 \cdots \forall X_k \exists X_{k+1} \cdots \exists X_n \Phi}$, Φ is 3CNF.

Recall the reduction from 3SAT to query containment $Q \subseteq Q'$:

- $\boldsymbol{Q}$ has 4 relations A, B, C, D each with 7 tuples.
- $\boldsymbol{Q'_\Phi}$ has one atom/clause. E.g. $(X_i \vee \neg X_j \vee X_k)$ becomes $B(x_i, x_k, x_j)$.
- $\exists X_1 \cdots \exists X_n \Phi$ iff $\exists h : Q'_\Phi \rightarrow Q$.

Proof of Π_2^p-Hardness

Reduction from $\forall 3CNF$: $\boxed{\Psi = \forall X_1 \cdots \forall X_k \exists X_{k+1} \cdots \exists X_n \Phi}$, Φ is 3CNF.

Recall the reduction from 3SAT to query containment $Q \subseteq Q'$:

- $\boldsymbol{Q}$ has 4 relations A, B, C, D each with 7 tuples.
- $\boldsymbol{Q'_\Phi}$ has one atom/clause. E.g. $(X_i \vee \neg X_j \vee X_k)$ becomes $B(x_i, x_k, x_j)$.
- $\exists X_1 \cdots \exists X_n \Phi$ iff $\exists h : Q'_\Phi \to Q$.

For each universal variable x_i, add the following atoms:

- Add $S(0, u_i, v_i) \wedge S(1, v_i, w_i) \wedge u_i < w_i$ to Q.
- Add $S(x_i, a_i, b_i) \wedge a_i < b_i$ to Q'_Φ.

$\boxed{Q \subseteq Q'_\Phi}$ holds iff both $x_i \mapsto 0$, $x_i \mapsto 1$ lead to a homomorphisms.

Summary

- The big question: what other extensions of CQ can we allow and still be able to decide containment?

- The following have been studied: inequalities, safe negation $\neg$, certain aggregates $\text{sum}, \text{min}, \text{max}, \text{count}$.

- The elegant containment/minimization theory for standard CQs quickly becomes very involved.

Chandra, A. K. and Merlin, P. M. (1977).
Optimal implementation of conjunctive queries in relational data bases.
In Hopcroft, J. E., Friedman, E. P., and Harrison, M. A., editors, *Proceedings of the 9th Annual ACM Symposium on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA*, pages 77–90. ACM.

Klug, A. C. (1988).
On conjunctive queries containing inequalities.
J. ACM, 35(1):146–160.

Kolaitis, P. G. and Vardi, M. Y. (1998).
Conjunctive-query containment and constraint satisfaction.
In Mendelzon, A. O. and Paredaens, J., editors, *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington, USA*, pages 205–213. ACM Press.

van der Meyden, R. (1997).
The complexity of querying indefinite data about linearly ordered domains.
J. Comput. Syst. Sci., 54(1):113–135.