GAV-sound with conjunctive queries

- Source and global schema as before:
  - source $R_1(A, B), R_2(B, C)$
  - Global schema: $T_1(A, C), T_2(B, C)$
- GAV mappings become sound:
  - $T_1 \supseteq \{x, y, z \mid R_1(x, y) \land R_2(y, z)\}$
  - $T_2 \supseteq R_2$
- Let $D_{\text{exact}}$ be the unique database that arises from the exact setting (with $\supseteq$ replaced by $=$)
- Then every database $D_{\text{sound}}$ that satisfies the sound setting also satisfies
  $$D_{\text{exact}} \subseteq D_{\text{sound}}$$
Conjunctive queries are monotone:

\[ D_1 \subseteq D_2 \quad \Rightarrow \quad Q(D_1) \subseteq Q(D_2) \]

Exact solution is a sound solution too, and is contained in every sound solution.

Hence certain answers for each conjunctive query

\[ \text{certain}(D, Q) = \bigcap_{D_{\text{sound}}} Q(D_{\text{sound}}) = Q(D_{\text{exact}}) \]

The solution for GAV-exact gives us certain answers for GAV-sound, for conjunctive (and more generally, monotone) queries.
Query answering using views

- General setting: database relations $R_1, \ldots, R_n$.
- Several views $V_1, \ldots, V_k$ are defined as results of queries over the $R_i$'s.
- We have a query $Q$ over $R_1, \ldots, R_n$.
- **Question**: Can $Q$ be answered in terms of the views?
  - In other words, can $Q$ be reformulated so it only refers to the data in $V_1, \ldots, V_k$?
Query answering using views in data integration

• LAV:
  ○ $R_1, \ldots, R_n$ are global schema relations; $Q$ is the global schema query
  ○ $V_i$’s are the sources defined over the global schema
  ○ We must answer $Q$ based on the sources (virtual integration)

• GAV:
  ○ $R_1, \ldots, R_n$ are the sources that are not fully available.
  ○ $Q$ is a query in terms of the source (or a query that was reformulated in terms of the sources)
  ○ Must see if it is answerable from the available views $V_1, \ldots, V_k$.

• We know the problem is impossible to solve for full relational algebra, hence we concentrate on conjunctive queries.
Conjunctive queries: rule-based notation

- We often write conjunctive queries as logical statements:

\[ \{ t, y, r \mid \exists d \ (\text{Movie}(t, d, y) \land \text{RV}(t, r) \land y > 2000) \} \]

- Rule-based:

\[ Q(t, y, r) :– \text{Movie}(t, d, y), \text{RV}(t, r), y > 2000 \]

- \( Q(t, y, r) \) is the head of the rule
- \( \text{Movie}(t, d, y), \text{RV}(t, r), y > 2000 \) is its body
- conjunctions are replaced by commas
- variables that occur in the body but not in the head (\( d \)) are assumed to be existentially quantified
- essentially logic programming notation (without functions)
Query answering using views: example

- Two relations in the database: \texttt{Cites(A,B)} (if A cites B), and \texttt{SameTopic(A,B)} (if A, B work on the same topic)
- Query $Q(x, y) :– \text{SameTopic}(x, y), \text{Cites}(x, y), \text{Cites}(y, x)$
- Two views are given:
  - $V_1(x, y) :– \text{Cites}(x, y), \text{Cites}(y, x)$
  - $V_2(x, y) :– \text{SameTopic}(x, y), \text{Cites}(x, x'), \text{Cites}(y, y')$
- Suggested rewriting: $Q'(x, y) :– V_1(x, y), V_2(x, y)$
- Why? Unfold using the definitions:
  $Q'(x, y) :– \text{Cites}(x, y), \text{Cites}(y, x), \text{SameTopic}(x, y), \text{Cites}(x, x'), \text{Cites}(y, y')$
- Equivalent to $Q$
Query answering using views

- Need a formal technique (algorithm): cannot rely on the semantics.
- Query $Q$:

  ```sql
  SELECT R1.A
  FROM R R1, R R2, S S1, S S2
       AND R1.B = 1 and S2.B = 1
  ```

- $Q(x) \leftarrow R(x, y), R(x, 1), S(x, z), S(x, 1)$
- Equivalent to $Q(x) \leftarrow R(x, 1), S(x, 1)$
- So if we have a view
  - $V(x, y) \leftarrow R(x, y), S(x, y)$ (i.e. $V = R \cap S$), then
  - $Q = \pi_A(\sigma_{B=1}(V))$
  - $Q$ can be rewritten (as a conjunctive query) in terms of $V$
Query rewriting

• Setting:
  ◦ Queries $V_1, \ldots, V_k$ over the same schema $\sigma$ (assume to be conjunctive; they define the views)
  ◦ Each $Q_i$ is of arity $n_i$
  ◦ A schema $\omega$ with relations of arities $n_1, \ldots, n_k$

• Given:
  ◦ a query $Q$ over $\sigma$
  ◦ a query $Q'$ over $\omega$

• $Q'$ is a rewriting of $Q$ if for every $\sigma$-database $D$,

$$Q(D) = Q'(V_1(D), \ldots, V_k(D))$$
Maximal rewriting

- Sometimes exact rewritings cannot be obtained
- \( Q' \) is a maximally-contained rewriting if:
  
  - it is contained in \( Q \):
    \[
    \forall D \quad Q'(V_1(D), \ldots, V_k(D)) \subseteq Q(D)
    \]
  
  - it is maximal such: if
    \[
    \forall D \quad Q''(V_1(D), \ldots, V_k(D)) \subseteq Q(D)
    \]
    
    for all \( D \), then
    \[
    Q'' \subseteq Q'
    \]
Side remark: query rewriting and certain answers

- If we have sources $R = (R_1, \ldots, R_k)$, we can view conditions
  \[
  V_1(D) = R_1, \ldots, V_k(D) = R_k
  \]
as an incomplete specification of a database $D$

- To answer $Q$ over $D$, given $R_1, \ldots, R_k$, we want to compute certain answers:
  \[
  \text{certain}(Q, R) = \bigcap \{Q(D) \mid V_1(D) = R_1, \ldots, V_k(D) = R_k\}
  \]

- If for every such $D$ we have $Q(D) = Q'(V_1(D), \ldots, V_k(D))$, then
  $\text{certain}(Q, R) = Q'$.

- But we may even look at a more general way of query answering by finding a rewriting $Q'$ so that
  \[
  \text{certain}(Q, R) = Q'(R)
  \]
Query rewriting: a naive algorithm

- Given:
  - conjunctive queries $V_1, \ldots, V_k$ over schema $\sigma$
  - a query $Q$ over $\sigma$
- Algorithm:
  - guess a query $Q'$ over the views
  - Unfold $Q'$ in terms of the views
  - Check if the unfolding is contained in $Q$
- If one unfolding is equivalent to $Q$, then $Q'$ is a rewriting
- Otherwise take the union of all unfoldings contained in $Q$
  - it is a maximally contained rewriting
Why is it not an algorithm yet?

- **Problem 1**: we do not yet know how to test containment and equivalence.
  - But we shall learn soon

- **Problem 2**: the guess stage.
  - There are infinitely many conjunctive queries.
  - We cannot check them all.
  - Solution: we only need to check a few.
Guessing rewritings

- A basic fact:
  - If there is a rewriting of $Q$ using $V_1, \ldots, V_k$, then there is a rewriting with no more conjuncts than in $Q$.
  - E.g., if $Q(x) :– R(x, y), R(x, 1), S(x, z), S(x, 1)$, we only need to check conjunctive queries over $V$ with at most 4 conjuncts.

- Moreover, maximally contained rewriting is obtained as the union of all conjunctive rewritings of length of $Q$ or less.

- Complexity: enumerate all candidates (exponentially many); for each an NP (or exponential) algorithm. Hence exponential time is required.

- Cannot lower this due to NP-completeness.
Containment and optimization of conjunctive queries

- Reminder:

  conjunctive queries
  = SPJ queries
  = rule-based queries
  = simple SELECT-FROM-WHERE SQL queries
  (only AND and equality in the WHERE clause)

- Extremely common, and thus special optimization techniques have been developed

- Reminder: for two relational algebra expressions $e_1, e_2$, $e_1 = e_2$ is undecidable.

- But for conjunctive queries, even $e_1 \subseteq e_2$ is decidable.

- Main goal of optimizing conjunctive queries: reduce the number of joins.
Optimization of conjunctive queries: an example

• Given a relation $R$ with two attributes $A, B$

• Two SQL queries:
  
  Q1
  
  SELECT R1.B, R1.A
  FROM R R1, R R2
  WHERE R2.A=R1.B

  Q2
  
  SELECT R3.A, R1.A
  FROM R R1, R R2, R R3

• Are they equivalent?

• If they are, we saved one join operation.

• In relational algebra:

  \[
  Q_1 = \pi_{2,1}(\sigma_{2=3}(R \times R))
  \]

  \[
  Q_2 = \pi_{5,1}(\sigma_{2=4 \land 4=5}(R \times R \times R))
  \]
Optimization of conjunctive queries cont’d

• Are \( Q_1 \) and \( Q_2 \) equivalent?
• If they are, we cannot show it by using equivalences for relational algebra expression.
• Because: they don’t decrease the number of \( \Join \) or \( \times \) operators, but \( Q_1 \) has 1 join, and \( Q_2 \) has 2.
• But \( Q_1 \) and \( Q_2 \) are equivalent. How can we show this?
• But representing queries as databases. This representation is very close to rule-based queries.

\[
Q_1(x, y) := R(y, x), R(x, z)
\]
\[
Q_2(x, y) := R(y, x), R(w, x), R(x, u)
\]
Conjunctive queries into tableaux

- Tableau: representing of a conjunctive query as a database
- We first consider queries over a single relation
- \( Q_1(x, y) :– R(y, x), R(x, z) \)
- \( Q_2(x, y) :– R(y, x), R(w, x), R(x, u) \)
- Tableaux:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>z</td>
</tr>
</tbody>
</table>

\( x \ y \leftarrow \) answer line

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>x</td>
</tr>
<tr>
<td>w</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>u</td>
</tr>
</tbody>
</table>

| x | y \leftarrow \) answer line

- Variables in the answer line are called distinguished
Tableau homomorphisms

- A homomorphism of two tableaux $f : T_1 \rightarrow T_2$ is a mapping
  
  \[ f : \{ \text{variables of } T_1 \} \rightarrow \{ \text{variables of } T_2 \} \cup \{ \text{constants} \} \]

- For every distinguished $x$, $f(x) = x$

- For every row $x_1, \ldots, x_k$ in $T_1$, $f(x_1), \ldots, f(x_k)$ is a row of $T_2$

- Query containment: $Q \subseteq Q'$ if $Q(D) \subseteq Q'(D)$ for every database $D$

- Homomorphism Theorem: Let $Q, Q'$ be two conjunctive queries, and $T, T'$ their tableaux. Then

  \[ Q \subseteq Q' \]
  
  if and only if
  
  there exists a homomorphism $f : T' \rightarrow T$
Applying the Homomorphism Theorem: $Q_1 = Q_2$

\[
\begin{array}{cc}
T1 & T2 \\
\hline
A & B \\
y & x \\
x & z \\
\hline
x & y \\
\end{array}
\begin{array}{cc}
A & B \\
y & x \\
w & x \\
\hline
x & u \\
x & y \\
\end{array}
\]

\[
f(x) = x, f(y) = y, f(z) = u, f(w) = y
\]

Hence $Q_1 \subseteq Q_2$

\[
\begin{array}{cc}
T1 & T2 \\
\hline
A & B \\
y & x \\
x & z \\
\hline
x & y \\
\end{array}
\begin{array}{cc}
A & B \\
y & x \\
w & x \\
\hline
x & u \\
x & y \\
\end{array}
\]

\[
f(x) = x, f(y) = y, f(z) = u
\]

Hence $Q_2 \subseteq Q1$
Applying the Homomorphism Theorem: Complexity

- Given two conjunctive queries, how hard is it to test if $Q_1 = Q_2$?
- it is easy to transform them into tableaux, from either SPJ relational algebra queries, or SQL queries, or rule-based queries
- But testing the existence of a homomorphism between two tableaux is hard: NP-complete. Thus, a polynomial algorithm is unlikely to exists.
- However, queries are small, and conjunctive query optimization is possible in practice.
Minimizing conjunctive queries

• Goal: given a conjunctive query $Q$, find an equivalent conjunctive query $Q'$ with the minimum number of joins.
• Assume $Q$ is
  \[ Q(\vec{x}) \leftarrow R_1(\vec{u}_1), \ldots, R_k(\vec{u}_k) \]
• Assume that there is an equivalent conjunctive query $Q'$ of the form
  \[ Q'(\vec{x}) \leftarrow S_1(\vec{v}_1), \ldots, S_l(\vec{v}_l) \]
  with $l < k$
• Then $Q$ is equivalent to a query of the form
  \[ Q'(\vec{x}) \leftarrow R_{i_1}(\vec{u}_{i_1}), \ldots, R_{i_l}(\vec{u}_{i_l}) \]
• In other words, to minimize a conjunctive query, one has to delete some subqueries on the right of :-
Minimizing conjunctive queries cont’d

- Given a conjunctive query $Q$, transform it into a tableau $T$.
- Let $Q'$ be a minimal conjunctive query equivalent to $Q$. Then its tableau $T'$ is a subset of $T$.
- Minimization algorithm:
  
  $$T' := T$$

  repeat until no change
  
  choose a row $t$ in $T'$
  
  if there is a homomorphism $f : T' \rightarrow T' - \{t\}$
  
  then $T' := T' - \{t\}$

  end

- Note: if there exists a homomorphism $T' \rightarrow T' - \{t\}$, then the queries defined by $T'$ and $T' - \{t\}$ are equivalent. Because: there is always a homomorphism from $T' - \{t\}$ to $T'$. (Why?)
Minimizing SPJ/conjunctive queries: example

- \( R \) with three attributes \( A, B, C \)
- SPJ query:
  \[
  Q = \pi_{AB}(\sigma_{B=4}(R)) \Join \pi_{BC}(\pi_{AB}(R) \Join \pi_{AC}(\sigma_{B=4}(R)))
  \]
- Equivalently, a SQL query:
  ```sql
  FROM R R1, R R2, R R3
  ```
- Translate into a conjunctive query:
  \[
  \exists x_1, z_1, z_2 (R(x, 4, z_1) \land R(x_1, 4, z_2) \land R(x_1, 4, z) \land y = 4)
  \]
- Rule-based:
  \[
  Q(x, y, z) :\neg R(x, 4, z_1), R(x_1, 4, z_2), R(x_1, 4, z), y = 4
  \]
Minimizing SPJ/conjunctive queries cont’d

- **Tableau** $T$:
  
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>4</td>
<td>$z_1$</td>
</tr>
<tr>
<td>$x_1$</td>
<td>4</td>
<td>$z_2$</td>
</tr>
<tr>
<td>$x_1$</td>
<td>4</td>
<td>$z$</td>
</tr>
<tr>
<td>$x$</td>
<td>4</td>
<td>$z$</td>
</tr>
</tbody>
</table>

- **Minimization, step 1**: is there a homomorphism from $T$ to
  
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>4</td>
<td>$z_2$</td>
</tr>
<tr>
<td>$x_1$</td>
<td>4</td>
<td>$z$</td>
</tr>
<tr>
<td>$x$</td>
<td>4</td>
<td>$z$</td>
</tr>
</tbody>
</table>

- **Answer**: No. For any homomorphism $f$, $f(x) = x$ (why?), thus the image of the first row is not in the small tableau.
Minimizing SPJ/conjunctive queries cont’d

• Step 2: Is \( T \) equivalent to

\[
\begin{array}{ccc}
A & B & C \\
x & 4 & z_1 \\
x_1 & 4 & z \\
x & 4 & z \\
\end{array}
\]

• Answer: Yes. Homomorphism \( f: f(z_2) = z \), all other variables stay the same.

• The new tableau is not equivalent to

\[
\begin{array}{ccc}
A & B & C \\
x & 4 & z_1 \\
x & 4 & z \\
\end{array}
\]

or

\[
\begin{array}{ccc}
A & B & C \\
x_1 & 4 & z \\
\end{array}
\]

• Because \( f(x) = x, f(z) = z \), and the image of one of the rows is not present.
Minimizing SPJ/conjunctive queries cont’d

- Minimal tableau:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>x</td>
<td>4</td>
<td>z₁</td>
</tr>
<tr>
<td>x₁</td>
<td>4</td>
<td>z</td>
</tr>
<tr>
<td>x</td>
<td>4</td>
<td>z</td>
</tr>
</tbody>
</table>

- Back to conjunctive query:

\[
Q'(x, y, z) \leftarrow R(x, y, z₁), R(x₁, y, z), y = 4
\]

- An SPJ query:

\[
\pi_{AB}(\sigma_{B=4}(R)) \bowtie \pi_{BC}(\sigma_{B=4}(R))
\]

  FROM R R₁, R R₂
Review of the journey

• We started with

\[ \pi_{AB}(\sigma_{B=4}(R)) \Join \pi_{BC}(\pi_{AB}(R) \Join \pi_{AC}(\sigma_{B=4}(R))) \]

• Translated into a conjunctive query
• Built a tableau and minimized it
• Translated back into conjunctive query and SPJ query
• Applied algebraic equivalences and obtained

\[ \pi_{AB}(\sigma_{B=4}(R)) \Join \pi_{BC}(\sigma_{B=4}(R)) \]

• Savings: one join.
All minimizations are equivalent

• Let $Q$ be a conjunctive query, and $Q_1$, $Q_2$ two conjunctive queries equivalent to $Q$.

• Assume that $Q_1$ and $Q_2$ are both minimal, and let $T_1$ and $T_2$ be their tableaux.

• Then $T_1$ and $T_2$ are isomorphic; that is, $T_2$ can be obtained from $T_1$ by renaming of variables.

• That is, all minimizations are equivalent.

• In particular, in the minimization algorithm, the order in which rows are considered, is irrelevant.
Equivalence of conjunctive queries: the general case

- So far we assumed that there is only one relation $R$, but what if there are many?
- Construct tableaux as before:
  
  \[
  Q(x, y): \neg B(x, y), R(y, z), R(y, w), R(w, y)
  \]

- Tableau:

<table>
<thead>
<tr>
<th>B:</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>y</td>
<td></td>
<td>y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R:</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>z</td>
<td>y</td>
<td>w</td>
</tr>
<tr>
<td>w</td>
<td>y</td>
<td>y</td>
</tr>
</tbody>
</table>

- Formally, a tableau is just a database where variables can appear in tuples, plus a set of distinguished variables.
Tableaux and multiple relations

• Given two tableaux $T_1$ and $T_2$ over the same set of relations, and the same distinguished variables, a homomorphism $h : T_1 \rightarrow T_2$ is a mapping
  \[ f : \{\text{variables of } T_1\} \rightarrow \{\text{variables of } T_2\} \]
such that
  - $f(x) = x$ for every distinguished variable, and
  - for each row $\vec{t}$ in $R$ in $T_1$, $f(\vec{t})$ is in $R$ in $T_2$.

• Homomorphism theorem: let $Q_1$ and $Q_2$ be conjunctive queries, and $T_1, T_2$ their tableaux. Then
  \[ Q_2 \subseteq Q_1 \]
  if and only if there exists a homomorphism $f : T_1 \rightarrow T_2$
Minimization with multiple relations

- The algorithm is the same as before, but one has to try rows in different relations. Consider homomorphism $f(z) = w$, and $f$ is the identity for other variables. Applying this to the tableau for $Q$ yields

\[
\begin{array}{c|c|c}
A & B & \hline
x & y \\
\end{array}
\quad
\begin{array}{c|c|c}
A & B & \hline
y & w \\
\end{array}
\quad
\begin{array}{c|c|c}
B: & A & \hline
x & y & \hline
\end{array}
\quad
\begin{array}{c|c|c}
R: & A & \hline
y & w & \hline
w & y & \hline
x & y & \hline
\end{array}
\]

- This cannot be further reduced, as for any homomorphism $f$, $f(x) = x$, $f(y) = y$.

- Thus $Q$ is equivalent to

\[
Q'(x, y) :\quad B(x, y), R(y, w), R(w, y)
\]

- One join is eliminated.
Query rewriting

• Recall the algorithm, for a given $Q$ and view definitions $V_1, \ldots, V_k$:
  ○ Look at all rewritings that have as at most as many joins as $Q$
  ○ check if they are contained in $Q$
  ○ take the union of those that are

• This is the maximally contained rewriting

• There are algorithms that prune the search space and make looking for rewritings contained in $Q$ more efficient
  ○ the bucket algorithm
  ○ MiniCon

• May see of them later
How hard is it to answer queries using views?

• Setting: we now have an actual content of the views.

• As before, a query is \( Q \) posed against \( D \), but must be answered using information in the views.

• Suppose \( I_1, \ldots, I_k \) are view instances. Two possibilities:
  - Exact mappings: \( I_j = V_j(D) \)
  - Sound mappings: \( I_j \subseteq V_j(D) \)

• We need certain answers for given \( \mathcal{I} = (I_1, \ldots, I_k) \):

\[
\text{certain}_{\text{exact}}(Q, \mathcal{I}) = \bigcap_{D: I_j = V_j(D) \text{ for all } j} Q(D)
\]

\[
\text{certain}_{\text{sound}}(Q, \mathcal{I}) = \bigcap_{D: I_j \subseteq V_j(D) \text{ for all } j} Q(D)
\]
How hard is it to answer queries using views?

- If certain\textsubscript{exact}(Q, I) or certain\textsubscript{sound}(Q, I) are impossible to obtain, we want maximally contained rewritings:
  - Q′(I) ⊆ certain\textsubscript{exact}(Q, I), and
  - if Q″(I) ⊆ certain\textsubscript{exact}(Q, I) then Q″(I) ⊆ Q′(I)
  - (and likewise for sound)

- How hard is it to compute this from I?

- In databases, we reason about complexity in two ways:
  - The big-O notation (O(n log n) vs O(n\(^2\)) vs O(2\(^n\)))
  - Complexity-theoretic notions: PTIME, NP, DLOGSPACE, etc

- Advantage of complexity-theoretic notions: if you have a O(2\(^n\)) algorithm, is it because the problem is inherently hard, or because we are not smart enough to come up with a better algorithm (or both)?
Complexity classes: what you always wanted to know but never dared to ask

- Or a 5/5-introduction: a five minute review that tells you what are likely to remember 5 years after taking a complexity theory course.
- The big divide: PTIME (computable in polynomial time, i.e. $O(n^k)$ for some fixed $k$)
- Inside PTIME: tractable queries (although high-degree polynomial are intractable)
- Outside PTIME: intractable queries (efficient algorithms are unlikely)
- Way outside PTIME: provably intractable queries (efficient algorithms do not exist)
  - EXPTIME: $c^n$-algorithms for a constant $c$. Could still be ok for not very large inputs
  - Even further – 2-EXPTIME: $c^{cn}$. Cannot be ok even for small inputs (compare $2^{10}$ and $2^{2^{10}}$).

L. Libkin
Inside PTIME

\[ AC^0 \subsetneq TC^0 \subseteq NC^1 \subseteq DLOG \subseteq NLOG \subseteq PTIME \]

- **AC^0**: very efficient parallel algorithms (constant time, simple circuits)
  - relational calculus
- **TC^0**: very efficient parallel algorithms in a more powerful computational model with counting gates
  - basic SQL (relational calculus/grouping/aggregation)
- **NC^1**: efficient parallel algorithms
  - regular languages
- **DLOG**: very little \( O(\log n) \) – space is required
  - SQL + (restricted) transitive closure
- **NLOG**: \( O(\log n) \) space is required if nondeterminism is allowed
  - SQL + transitive closure (as in the SQL3 standard)
Beyond PTIME

\[
\text{PTIME} \subseteq \left\{ \begin{array}{c}
\text{NP} \\
\text{coNP}
\end{array} \right\} \subseteq \text{PSPACE}
\]

- **PTIME**: can solve a problem in polynomial time
- **NP**: can check a given candidate solution in polynomial time
  - another way of looking at it: guess a solution, and then verify if you guessed it right in polynomial time
- **coNP**: complement of NP – verify that all “reasonable” candidates are solutions to a given problem.
  - Appears to be harder than NP but the precise relationship isn’t known
- **PSPACE**: can be solved using memory of polynomial size (but perhaps an exponential-time algorithm)
Complete problems

- These are the hardest problems in a class.
- If our problem is as hard as a complete problem, it is very unlikely it can be done with lower complexity.
- For NP:
  - SAT (satisfiability of Boolean formulae)
  - many graph problems (e.g. 3-colourability)
  - Integer linear programming etc
- For PSPACE:
  - Quantified SAT
  - Two XML DTDs are equivalent
Complexity of query answering

- We want the complexity of finding
  \[ \text{certain}_{\text{exact}}(Q, \mathcal{I}) \quad \text{or} \quad \text{certain}_{\text{sound}}(Q, \mathcal{I}) \]
  in terms of the size of \( \mathcal{I} \)
- If all view definitions are conjunctive queries and \( Q \) is a relational algebra or a SQL query, then the complexity is \( \text{coNP} \).
- (blackboard)
- This is too high!
- If all view definitions are conjunctive queries and \( Q \) is a conjunctive query, then the complexity is \( \text{PTIME} \).
  - Because: the maximally contained rewriting computes certain answers!
## Complexity of query answering

<table>
<thead>
<tr>
<th>view language</th>
<th>CQ</th>
<th>CQ ≠</th>
<th>relational calculus</th>
</tr>
</thead>
<tbody>
<tr>
<td>CQ</td>
<td>ptime</td>
<td>coNP</td>
<td>undecidable</td>
</tr>
<tr>
<td>CQ ≠</td>
<td>ptime</td>
<td>coNP</td>
<td>undecidable</td>
</tr>
<tr>
<td>relational calculus</td>
<td>undecidable</td>
<td>undecidable</td>
<td>undecidable</td>
</tr>
</tbody>
</table>

**CQ** – conjunctive queries

**CQ ≠** – conjunctive queries with **inequalities**

(for example, \( Q(x) :− R(x, y), S(y, z), x \neq z \) )
Complexity of query answering: coNP-completeness idea

- Start with a graph $G$ – this is our instance
- $D$ is $G$ together with a colouring, with 3 colours; each node is assigned one colour.
- $Q$ asks if we have an edge $(a, b)$ with $a \neq b$ and $a, b$ of the same colour.
- If $G$ is not 3-colourable, then every instance $D$ would satisfy $Q$
- Otherwise, if $G$ is 3-colourable, we can find extensions that are and that are not 3-colourable – hence certain answers are empty.
- Thus if we can compute certain answers, we can test non-3-colourability $\Rightarrow$ coNP-completeness.