Data Exchange:
Source instance ⇒ Target instance

Problem 1:
▶ There may be infinitely many valid target instances for a given source instance

Problem 2. Query Answering
▶ What does it mean to answer a query over the target schema?
▶ Can we answer queries using only one target instance?

Fagin, Kolaitis, Miller, Popa, 2003:
▶ Use a certain answers semantics
▶ Canonical Solution: "good" target instance that can be computed in polynomial time
▶ Union of conjunctive queries: their certain answers can be computed using only the canonical solution
Data Exchange:
Source instance ⇒ Target instance

Problem 1:

- There may be infinitely many valid target instances for a given source instance
Data Exchange:
Source instance ⇒ Target instance

Problem 1:
- There may be infinitely many valid target instances for a given source instance

Problem 2. Query Answering
Data Exchange:
Source instance ⇒ Target instance

Problem 1:
- There may be infinitely many valid target instances for a given source instance

Problem 2. Query Answering
- What does it mean to answer a query over the target schema?
- Can we answer queries using only one target instance?
Data Exchange:  
Source instance ⇒ Target instance

Problem 1:
- There may be infinitely many valid target instances for a given source instance

Problem 2. Query Answering
- What does it mean to answer a query over the target schema?
- Can we answer queries using only one target instance?

Fagin, Kolaitis, Miller, Popa, 2003:
- Use a certain answers semantics
- Canonical Solution: “good” target instance that can be computed in polynomial time
- Union of conjunctive queries: their certain answers can be computed using only the canonical solution
We propose a tractable query language that express *negation*

For union of conjunctive queries, the certain answers can be computed in polynomial time.

- Union of conjunctive queries have this good property because they are preserved under homomorphisms
We propose a tractable query language that express *negation*

For union of conjunctive queries, the certain answers can be computed in polynomial time.

- Union of conjunctive queries have this good property because they are preserved under homomorphisms
- **Datalog** queries can also be computed in polynomial time
We propose a tractable query language that express *negation*

For union of conjunctive queries, the certain answers can be computed in polynomial time.

- Union of conjunctive queries have this good property because they are preserved under homomorphisms
- *Datalog* queries can also be computed in polynomial time
- Both *Datalog* and union of conjunctive queries keep us on the realm of positive

Computing certain answers of conjunctive queries with inequalities is *coNP*-complete

How can we add negation while keeping good properties for data exchange?
We propose a tractable query language that expresses *negation*. For union of conjunctive queries, the certain answers can be computed in polynomial time.

- Union of conjunctive queries have this good property because they are preserved under homomorphisms.
- Datalog queries can also be computed in polynomial time.
- Both Datalog and union of conjunctive queries keep us on the realm of positive.
- Computing certain answers of conjunctive queries with inequalities is coNP-complete.
We propose a tractable query language that express *negation*

For union of conjunctive queries, the certain answers can be computed in polynomial time.

- Union of conjunctive queries have this good property because they are preserved under homomorphisms
- **Datalog** queries can also be computed in polynomial time
- Both **Datalog** and union of conjunctive queries keep us on the realm of positive

- Computing certain answers of conjunctive queries with inequalities is $\text{coNP}$-complete

How can we add negation while keeping good properties for data exchange?
Query Languages for Data Exchange: Beyond Unions of Conjunctive Queries

Marcelo Arenas  Pablo Barceló  Juan Reutter
PUC Chile  Univ. of Chile  PUC Chile

Khipu: South Andean Center for Database Research
Data exchange settings:

- Source schema $S$ (Source instances with constant values)
- Target schema $T$ (Target instance can contain nulls)
- Set $\Sigma_{st}$ of $st$-tgds of the form:

$$\phi_S(\bar{x}) \rightarrow \exists \bar{y} \psi_T(\bar{x}, \bar{y})$$

- $C(a)$ holds if $a$ is a constant value

An instance $J$ is a solution for $I$ if

- $(I, J) \models \Sigma_{st}$
A homomorphism from $J_1$ to $J_2$ is a function that:

▶ Preserve the relations
▶ Is the identity on constants

$J$ is a universal solution if

▶ There is a homomorphism from $J$ to every other solution
Homomorphism and Universal Solutions

A homomorphism from $J_1$ to $J_2$ is a function that:

▶ Preserve the relations
▶ Is the identity on constants

$J$ is a universal solution if

▶ There is a homomorphism from $J$ to every other solution

Canonical universal solution can be computed in polynomial time using a chase procedure (FKMP 03).
Certain answers for conjunctive queries with negation are empty/false

Example:

\[ M : \]
\[ G(x, y) \rightarrow E(x, y) \]
\[ S(x) \rightarrow P(x) \]
\[ T(x) \rightarrow R(x) \]

\[ Q : \exists x \exists y \exists z (E(x, z) \land E(z, y) \land \neg E(x, y)) \]
Certain answers for conjunctive queries with negation are empty/false

Example:

\[ M : \begin{align*} G(x, y) & \rightarrow E(x, y) \\ S(x) & \rightarrow P(x) \\ T(x) & \rightarrow R(x) \end{align*} \]

\[ Q : \exists x \exists y \exists z (E(x, z) \land E(z, y) \land \neg E(x, y)) \]

\[ J_1 : \begin{align*} E(a, b) \\ E(b, c) \end{align*} \]
Certain answers for conjunctive queries with negation are empty/false

Example:

\[ M : \begin{align*}
G(x, y) & \rightarrow E(x, y) \\
S(x) & \rightarrow P(x) \\
T(x) & \rightarrow R(x)
\end{align*} \]

\[ Q : \exists x \exists y \exists z \left( E(x, z) \land E(z, y) \land \neg E(x, y) \right) \]

\[ J_2 : \\
E(a, b) \\
E(b, c) \\
E(a, c) \]
Certain answers for conjunctive queries with negation are empty/false

Example:

\[ M : \]
\[ G(x, y) \rightarrow E(x, y) \]
\[ S(x) \rightarrow P(x) \]
\[ T(x) \rightarrow R(x) \]

\[ Q : \exists x \exists y \exists z (E(x, z) \land E(z, y) \land \neg E(x, y)) \]

\[ J_2 : \]
\[ E(a, b) \]
\[ E(b, c) \]
\[ E(a, c) \]

\[ J_2 \text{ is also a solution!} \]
Certain answers for conjunctive queries with negation are empty/false

Example:

\[ M : \]
\[ G(x, y) \rightarrow E(x, y) \]
\[ S(x) \rightarrow P(x) \]
\[ T(x) \rightarrow R(x) \]

\[ Q : \]
\[ \exists x \exists y \exists z (E(x, z) \land E(z, y) \land \neg E(x, y)) \]

\[ J_2 : \]
\[ E(a, b) \]
\[ E(b, c) \]
\[ E(a, c) \]

- Idea: solution where \( E \) contains the transitive closure of \( G \)
Certain answers for conjunctive queries with negation are empty/false

Example:

\[\mathcal{M} : \quad G(x, y) \rightarrow E(x, y)\]
\[S(x) \rightarrow P(x)\]
\[T(x) \rightarrow R(x)\]

\[Q : \quad \exists x \exists y \exists z (E(x, z) \land E(z, y) \land \neg E(x, y))\]

\[J_2 : \]
\[E(a, b)\]
\[E(b, c)\]
\[E(a, c)\]

- Idea: solution where \( E \) contains the transitive closure of \( G \)
- \( Q \) is always false in that solution!
Unions of *positive* queries and conjunctive queries with negation are much more interesting

Example:

\[ M : \begin{align*}
G(x, y) & \rightarrow E(x, y) \\
S(x) & \rightarrow P(x) \\
T(x) & \rightarrow R(x)
\end{align*} \]

\[ Q : \begin{align*}
\exists x \exists y (P(x) \land R(y) \land E(x, y)) & \lor \\
\exists x \exists y \exists z (E(x, z) \land E(z, y) \land \neg E(x, y))
\end{align*} \]
Unions of *positive* queries and conjunctive queries with negation are much more interesting

Example:

\[
\begin{align*}
M : & \quad G(x, y) \rightarrow E(x, y) \\
    & \quad S(x) \rightarrow P(x) \\
    & \quad T(x) \rightarrow R(x)
\end{align*}
\]

\[
Q : \quad \exists x \exists y (P(x) \land R(y) \land E(x, y)) \lor \\
\quad \exists x \exists y \exists z (E(x, z) \land E(z, y) \land \neg E(x, y))
\]

▶ If we try to falsify the second disjunct (computing the transitive closure of \(G\)), we may end up satisfying the first one.
Unions of *positive* queries and conjunctive queries with negation are much more interesting

Example:

\[ M : \]
\[
G(x, y) \rightarrow E(x, y) \\
S(x) \rightarrow P(x) \\
T(x) \rightarrow R(x)
\]

\[ Q : \]
\[
\exists x \exists y (P(x) \land R(y) \land E(x, y)) \lor \\
\exists x \exists y \exists z (E(x, z) \land E(z, y) \land \neg E(x, y))
\]

▶ If we try to falsify the second disjunct (computing the transitive closure of \( G \)), we may end up satisfying the first one.

▶ \( Q \) holds if there exist \( a, b \):
  ▶ \( P(a), R(b) \) hold
  ▶ \((a, b)\) is in the transitive closure of \( G \)
Using **Datalog** we compute certain answers for queries with negation in polynomial time

Idea: Encode $Q$ using **Datalog** programs

$M : \begin{align*} G(x, y) \rightarrow E(x, y) \\
S(x) \rightarrow P(x) \\
T(x) \rightarrow R(x) \end{align*}$

$Q : \begin{align*} \exists x \exists y (P(x) \land R(y) \land E(x, y)) \lor \\
\exists x \exists y \exists z (E(x, z) \land E(z, y) \land \neg E(x, y)) \end{align*}$
Using **Datalog** we compute certain answers for queries with negation in polynomial time

Idea: Encode $Q$ using **Datalog** programs

$M :$

- $G(x, y) \rightarrow E(x, y)$
- $S(x) \rightarrow P(x)$
- $T(x) \rightarrow R(x)$

$Q :$

- $\exists x \exists y (P(x) \land R(y) \land E(x, y)) \lor$
- $\exists x \exists y \exists z (E(x, z) \land E(z, y) \land \neg E(x, y))$

- $S(x, y) \leftarrow E(x, y)$
- $S(x, y) \leftarrow S(x, z), S(z, y)$
- $true \leftarrow P(x), R(y), S(x, y)$
Using \textbf{Datalog} we compute certain answers for queries with negation in polynomial time

Idea: Encode $Q$ using \textbf{Datalog} programs

\begin{align*}
  \mathcal{M} : & \quad G(x, y) \rightarrow E(x, y) \\
  & \quad S(x) \rightarrow P(x) \\
  & \quad T(x) \rightarrow R(x)
\end{align*}

\begin{align*}
  Q : & \quad \exists x \exists y (P(x) \land R(y) \land E(x, y)) \lor \\
  & \quad \exists x \exists y \exists z (E(x, z) \land E(z, y) \land \neg E(x, y))
\end{align*}

\begin{align*}
  S(x, y) & \leftarrow E(x, y) \\
  S(x, y) & \leftarrow S(x, z), \; S(z, y) \\
  \text{true} & \leftarrow P(x), \; R(y), \; S(x, y)
\end{align*}

We only evaluate this program in the canonical solution.
Queries with inequalities cannot be answered directly in universal solutions

Problem:
We cannot add inequalities directly to Datalog.

- Preservation under homomorphisms is lost
- Language becomes intractable (Abiteboul, Dushka 1998)

Homomorphisms in data exchange are the identity on constants
- Thus, inequalities witnessed by constants are preserved under homomorphisms
Contributions

Query Language that extends \textsc{Datalog} with negation

- As good as \textsc{Datalog} for data exchange
- Can be used to find new tractable classes of queries

...And further

- Combined complexity of the new language and related query languages
Outline

Formalization
- \( \text{DATALOG}^{c(\neq)} \) programs

Beyond union of conjunctive queries
- Expressive power of \( \text{DATALOG}^{c(\neq)} \)
- New tractable classes of queries

Combined Complexity
- \( \text{DATALOG}^{c(\neq)} \) and queries with inequalities
- Restricting to \( \text{LAV} \) settings

Concluding remarks
**Datalog**ₚ programs extend **Datalog** with inequalities over constants

**Definition:**
A collection of constant-inequality rules of the form:

\[ S(\bar{x}) \leftarrow \ldots \]

- predicate symbols
- variables under predicate **C**
- inequalities of the form \( u \neq v \),
  \( u \) and \( v \) must be under predicate **C**
\textsc{Datalog}^{C(\neq)} \text{ programs extend Datalog with inequalities over constants}

**Definition:**
A collection of constant-inequality rules of the form:

\[ S(\bar{x}) \leftarrow \ldots \]

- predicate symbols
- variables under predicate \( C \)
- inequalities of the form \( u \neq v \),
  \( u \) and \( v \) must be under predicate \( C \)

**Example:**

\[
\begin{align*}
S(x, y) & \leftarrow E(x, y) \\
S(x, y) & \leftarrow S(x, z), S(z, y), C(x), C(z), C(y), x \neq z, y \neq z \\
true & \leftarrow P(x), R(y), S(x, y), C(x), C(y), x \neq y
\end{align*}
\]
**Datalog**\(^C(\neq)\) programs extend **Datalog** with inequalities over constants

**Definition:**
A collection of constant-inequality rules of the form:

\[
S(\bar{x}) \leftarrow ... \\
\quad \text{predicate symbols} \\
\quad \text{variables under predicate } C \\
\quad \text{inequalities of the form } u \neq v, \\
\quad \text{ } u \text{ and } v \text{ must be under predicate } C
\]

**Example:**

\[
S(x, y) \leftarrow E(x, y) \\
S(x, y) \leftarrow S(x, z), S(z, y), C(x), C(z), C(y), x \neq z, y \neq z \\
true \leftarrow P(x), R(y), S(x, y), C(x), C(y), x \neq y
\]
**Datalog**\(^C(\neq)\) programs extend **Datalog** with inequalities over constants

**Definition:**
A collection of constant-inequality rules of the form:

\[ S(\bar{x}) \leftarrow \ldots \]

- predicate symbols
- variables under predicate \( C \)
- inequalities of the form \( u \neq v \),
  \( u \) and \( v \) must be under predicate \( C \)

**Example:**

\[
\begin{align*}
S(x, y) & \leftarrow E(x, y) \\
S(x, y) & \leftarrow S(x, z), S(z, y), C(x), C(z), C(y), x \neq z, y \neq z \\
true & \leftarrow P(x), R(y), S(x, y), C(x), C(y), x \neq y
\end{align*}
\]
**Datalog**\(^\mathcal{C}(\neq)\) programs extend **Datalog** with inequalities over constants

**Definition:**
A collection of constant-inequality rules of the form:

\[ S(\bar{x}) \leftarrow \ldots \]

- predicate symbols
- variables under predicate \(C\)
- inequalities of the form \(u \neq v\),
  
  \(u\) and \(v\) must be under predicate \(C\)

**Example:**

\[
\begin{align*}
S(x, y) & \leftarrow E(x, y) \\
S(x, y) & \leftarrow S(x, z), S(z, y), C(x), C(z), C(y), x \neq z, y \neq z \\
\text{true} & \leftarrow P(x), R(y), S(x, y), C(x), C(y), x \neq y
\end{align*}
\]
\textbf{Datalog}^C(\neq) programs have the same good properties as conjunctive queries

\begin{itemize}
  \item \textbf{Datalog}^C(\neq) programs are preserved under homomorphisms
\end{itemize}
\textbf{Datalog}^C(\neq) programs have the same good properties as conjunctive queries

- \textbf{Datalog}^C(\neq) programs are preserved under homomorphisms
  - \textbf{Datalog} programs are preserved under homomorphisms
  - every inequality must be witnessed by constants
  - homomorphisms are the identity on constants

\textbf{Proposition} Certain answers of \textbf{Datalog}^C(\neq) programs can be computed by evaluating the programs over the canonical universal solution.

\textbf{Theorem} Computing the certain answers of a \textbf{Datalog}^C(\neq) program takes polynomial time (data complexity)
**Datalog**\(^C(\neq)\) programs have the same good properties as conjunctive queries

- **Datalog**\(^C(\neq)\) programs are preserved under homomorphisms
  - **Datalog** programs are preserved under homomorphisms
  - every inequality must be witnessed by constants
  - homomorphisms are the identity on constants

**Proposition**

Certain answers of **Datalog**\(^C(\neq)\) programs can be computed by evaluating the programs over the canonical universal solution.
**Datalog**\(^C(\neq)\) programs have the same good properties as conjunctive queries

- **Datalog**\(^C(\neq)\) programs are preserved under homomorphisms
  - **Datalog** programs are preserved under homomorphisms
  - every inequality must be witnessed by constants
  - homomorphisms are the identity on constants

**Proposition**

Certain answers of **Datalog**\(^C(\neq)\) programs can be computed by evaluating the programs over the canonical universal solution.

**Theorem**

Computing the certain answers of a **Datalog**\(^C(\neq)\) program takes polynomial time (data complexity)
$\text{DATALOG}_c^C(\neq)$ can express queries with negation

**Theorem**

Every union of conjunctive query with at most
- One negated atom
- One inequality

per disjunct, can be expressed as a $\text{DATALOG}_c^C(\neq)$ program.
\textbf{DATALOG}_C(\neq) \text{ can express queries with negation}

\begin{center}
\textbf{Theorem}

Every union of conjunctive query with at most
\begin{itemize}
  \item One negated atom
  \item One inequality
\end{itemize}
per disjunct, can be expressed as a \textbf{DATALOG}_C(\neq) program.
\begin{itemize}
  \item Certain answers for this class of queries can be computed in polynomial time
  \item Result for inequalities had been proved by FKMP03 using different techniques
\end{itemize}
\end{center}
\( \text{Datalog}^{\text{C}(\neq)} \) can express queries with negation.

**Theorem**

Every union of conjunctive query with at most
- One negated atom
- One inequality per disjunct, can be expressed as a \( \text{Datalog}^{\text{C}(\neq)} \) program.

- Certain answers for this class of queries can be computed in polynomial time
- Result for inequalities had been proved by FKMP03 using different techniques
- Next example gives a hint on the proof
Writing DATALOG\(^C(\neq)\) programs to answer queries with negation

\[
Q : \quad \exists x \exists y (E(x, y) \land x \neq y) \lor \\
\exists x \exists y \exists z (E(x, y) \land E(y, z) \land \neg E(x, z))
\]
Writing \( \text{DATALOG}^{C(\neq)} \) programs to answer queries with negation

\[
Q : \quad \exists x \exists y \ (E(x, y) \land x \neq y) \lor \\
\exists x \exists y \exists z \ (E(x, y) \land E(y, z) \land \neg E(x, z))
\]

\[
\begin{align*}
dom(x) & \leftarrow E(x, z) \\
dom(x) & \leftarrow E(z, x)
\end{align*}
\]

- Collect the domain
Writing $\textsc{Datalog}^{C(\neq)}$ programs to answer queries with negation

$$Q : \quad \exists x \exists y \ (E(x, y) \land x \neq y) \lor \\
\quad \exists x \exists y \exists z \ (E(x, y) \land E(y, z) \land \neg E(x, z))$$

- $\text{dom}(x) \leftarrow E(x, z)$
- $\text{dom}(x) \leftarrow E(z, x)$  - Collect the domain
- $\text{EQ}(x, x) \leftarrow \text{dom}(x)$  - Formalize the Equality
- $\text{EQ}(x, y) \leftarrow \text{EQ}(x, w), \text{EQ}(w, y)$
Writing \( \text{DATALOG}^{C(\neq)} \) programs to answer queries with negation

\[
Q : \quad \begin{align*}
\exists x \exists y & \ (E(x, y) \land x \neq y) \lor \\
\exists x \exists y \exists z & \ (E(x, y) \land E(y, z) \land \neg E(x, z))
\end{align*}
\]

\[
\begin{align*}
\text{dom}(x) & \leftarrow E(x, z) \\
\text{dom}(x) & \leftarrow E(z, x) \quad \text{- Collect the domain} \\
\text{EQ}(x, x) & \leftarrow \text{dom}(x) \quad \text{- Formalize the Equality} \\
\text{EQ}(x, y) & \leftarrow \text{EQ}(x, w), \text{EQ}(w, y) \quad \text{- Copy E into U} \\
U(x, y) & \leftarrow E(x, y)
\end{align*}
\]
Writing \textsc{Datalog}^C(\neq) programs to answer queries with negation

\begin{align*}
Q : & \quad \exists x \exists y \ (E(x, y) \land x \neq y) \lor \\
& \quad \exists x \exists y \exists z \ (E(x, y) \land E(y, z) \land \neg E(x, z))
\end{align*}

\begin{align*}
\text{dom}(x) & \leftarrow E(x, z) \\
\text{dom}(x) & \leftarrow E(z, x) \\
\text{EQ}(x, x) & \leftarrow \text{dom}(x) \quad \text{- Collect the domain} \\
\text{EQ}(x, y) & \leftarrow \text{EQ}(x, w), \text{EQ}(w, y) \quad \text{- Formalize the Equality} \\
U(x, y) & \leftarrow E(x, y) \quad \text{- Copy E into U} \\
U(x, y) & \leftarrow \text{EQ}(u, v), \text{EQ}(u, x), \text{EQ}(v, y) \quad \text{- Replace equals in U}
\end{align*}
Writing $\text{Datalog}^C(\neq)$ programs to answer queries with negation

$$Q : \quad \exists x \exists y (E(x, y) \land x \neq y) \lor$$
$$\exists x \exists y \exists z (E(x, y) \land E(y, z) \land \neg E(x, z))$$

- Collect the domain
- Formalize the Equality
- Copy E into U
- Replace equals in U
- Simulate negation

$$\begin{align*}
dom(x) & \leftarrow E(x, z) \\
dom(x) & \leftarrow E(z, x) \\
EQ(x, x) & \leftarrow dom(x) \\
EQ(x, y) & \leftarrow EQ(x, w), EQ(w, y) \\
U(x, y) & \leftarrow E(x, y) \\
U(x, y) & \leftarrow EQ(u, v), EQ(u, x), EQ(v, y) \\
U(x, y) & \leftarrow U(x, z), U(z, y)
\end{align*}$$
Writing **Datalog**\(^C(\neq)\) programs to answer queries with negation

\[
Q : \quad \exists x \exists y (E(x, y) \land x \neq y) \lor \\
\exists x \exists y \exists z (E(x, y) \land E(y, z) \land \neg E(x, z))
\]

\[
\begin{align*}
\text{dom}(x) & \leftarrow E(x, z) \\
\text{dom}(x) & \leftarrow E(z, x) \\
EQ(x, x) & \leftarrow \text{dom}(x) & \text{- Collect the domain} \\
EQ(x, y) & \leftarrow EQ(x, w), EQ(w, y) & \text{- Formalize the Equality} \\
U(x, y) & \leftarrow E(x, y) & \text{- Copy E into U} \\
U(x, y) & \leftarrow EQ(u, v), \\
& \quad EQ(u, x), EQ(v, y) & \text{- Replace equals in U} \\
U(x, y) & \leftarrow U(x, z), U(z, y) & \text{- Simulate negation} \\
EQ(x, y) & \leftarrow U(x, y) & \text{- Simulate inequality}
\end{align*}
\]
Writing \textbf{Datalog} \textsuperscript{C(\neq)} programs to answer queries with negation

\[ Q : \quad \exists x \exists y \ (E(x, y) \land x \neq y) \lor \exists x \exists y \exists z \ (E(x, y) \land E(y, z) \land \neg E(x, z)) \]

<table>
<thead>
<tr>
<th>Rule</th>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{dom}(x) &amp; \leftarrow E(x, z)</td>
<td></td>
</tr>
<tr>
<td>\texttt{dom}(x) &amp; \leftarrow E(z, x)</td>
<td></td>
</tr>
<tr>
<td>\texttt{EQ}(x, x) &amp; \leftarrow \texttt{dom}(x)</td>
<td>- Collect the domain</td>
</tr>
<tr>
<td>\texttt{EQ}(x, y) &amp; \leftarrow \texttt{EQ}(x, w), \texttt{EQ}(w, y)</td>
<td>- Formalize the Equality</td>
</tr>
<tr>
<td>\texttt{U}(x, y) &amp; \leftarrow E(x, y)</td>
<td>- Copy E into U</td>
</tr>
<tr>
<td>\texttt{U}(x, y) &amp; \leftarrow \texttt{EQ}(u, v), \texttt{EQ}(u, x), \texttt{EQ}(v, y)</td>
<td>- Replace equals in U</td>
</tr>
<tr>
<td>\texttt{U}(x, y) &amp; \leftarrow \texttt{U}(x, z), \texttt{U}(z, y)</td>
<td>- Simulate negation</td>
</tr>
<tr>
<td>\texttt{EQ}(x, y) &amp; \leftarrow \texttt{U}(x, y)</td>
<td>- Simulate inequality</td>
</tr>
<tr>
<td>\texttt{TRUE} &amp; \leftarrow \texttt{EQ}(z, y), \texttt{C}(y), \texttt{C}(z), y \neq z</td>
<td>- Answer</td>
</tr>
</tbody>
</table>
Outline

Formalization

- $\text{DATALOG}^C(\neq)$ programs

Beyond union of conjunctive queries

- Expressive power of $\text{DATALOG}^C(\neq)$
- New tractable classes of queries

Combined Complexity

- $\text{DATALOG}^C(\neq)$ and queries with inequalities
- Restricting to LAV settings

Concluding remarks
Classes of queries

\((UCQ)CQ\)
- (union) of conjunctive queries

\((UCQ\neq)CQ\neq\)
- (union) of conjunctive queries with inequalities

\(k-CQ\neq\)
- conjunctive queries with at most \(k\) inequalities
Certain answers for conjunctive queries with two inequalities is intractable (data complexity)

[Madry 05]:

- The certain answers problem is $\text{coNP}$-complete for $2$-$\text{CQ} \neq$
Certain answers for conjunctive queries with two inequalities is intractable (data complexity)

[Madry 05]:
- The certain answers problem is $\text{coNP}$-complete for $2\text{-CQ} \neq$

We find an interesting tractable fragment for this class of queries, using translation into $\text{DATALOG}^{\text{C}(\neq)}$ programs
We need to define two restrictions

- Constant Joins
- Almost constant inequalities
We need to define two restrictions

- Constant Joins
- Almost constant inequalities

**Constant Joins:**
No null values can witness a join of a relation
We need to define two restrictions

- **Constant Joins**
- **Almost constant inequalities**

**Constant Joins:**
No null values can witness a join of a relation

\[ M : \]

\[ P(u, v) \rightarrow T(u, v) \]
\[ Q(u, v) \rightarrow \exists w U(u, w) \]

\[ Q_1 : \]
\[ \exists x \exists y \exists z (T(x, y) \land U(x, z)) \]
\[ Q_2 : \]
\[ \exists x \exists y \exists z (U(x, z) \land U(y, z)) \]
We need to define two restrictions

- **Constant Joins**
- **Almost constant inequalities**

**Constant Joins:**
No null values can witness a join of a relation

\[ M : \quad P(u, v) \rightarrow T(u, v) \]
\[ Q(u, v) \rightarrow \exists w U(u, w) \]

\[ Q_1 : \quad \exists x \exists y \exists z (T(x, y) \land U(x, z)) \quad \text{YES} \]
\[ Q_2 : \quad \exists x \exists y \exists z (U(x, z) \land U(y, z)) \]
We need to define two restrictions

- **Constant Joins**
- **Almost constant inequalities**

**Constant Joins:**
No null values can witness a join of a relation

\[
\mathcal{M} : \\
P(u, v) \rightarrow T(u, v) \\
Q(u, v) \rightarrow \exists w U(u, w)
\]

\[
Q_1 : \exists x \exists y \exists z (T(x, y) \land U(x, z)) \quad \text{YES}
\]

\[
Q_2 : \exists x \exists y \exists z (U(x, z) \land U(y, z)) \quad \text{NO}
\]
We need to define two restrictions

- Constant Joins
- Almost constant inequalities

Almost constant inequalities:
Every inequality can be witnessed by at most 1 null value
We need to define two restrictions

- Constant Joins
- Almost constant inequalities

Almost constant inequalities:
Every inequality can be witnessed by at most 1 null value

\[ M : \]
\[ P(u, v) \rightarrow T(u, v) \]
\[ Q(u, v) \rightarrow \exists w U(u, w) \]

\[ Q_1 : \exists x \exists y \exists z (U(x, y) \land U(x, z) \land x \neq z) \]
\[ Q_2 : \exists x \exists y \exists z (U(x, y) \land U(x, z) \land y \neq z) \]
We need to define two restrictions

- Constant Joins
- Almost constant inequalities

Almost constant inequalities:
Every inequality can be witnessed by at most 1 null value

\[ M : \ P(u, v) \rightarrow T(u, v) \]
\[ Q(u, v) \rightarrow \exists w U(u, w) \]

\[ Q_1 : \exists x \exists y \exists z (U(x, y) \land U(x, z) \land x \neq z) \text{ YES} \]

\[ Q_2 : \exists x \exists y \exists z (U(x, y) \land U(x, z) \land y \neq z) \]
We need to define two restrictions

- Constant Joins
- Almost constant inequalities

Almost constant inequalities:
Every inequality can be witnessed by at most 1 null value

\[ M : \]
\[ P(u, v) \rightarrow T(u, v) \]
\[ Q(u, v) \rightarrow \exists w U(u, w) \]

\[ Q_1 : \exists x \exists y \exists z (U(x, y) \land U(x, z) \land x \neq z) \quad \text{YES} \]
\[ Q_2 : \exists x \exists y \exists z (U(x, y) \land U(x, z) \land y \neq z) \quad \text{NO} \]
We use $\text{DATALOG}^C(\neq)$ to find a tractable fragment for union of conjunctive queries with at most two inequalities.

**Theorem**

Every $2$-$\text{UCQ} \neq$ with:
- constant joins
- almost constant inequalities

... can be expressed as a $\text{DATALOG}^C(\neq)$ program in data exchange. 

Removing any one of these conditions yields intractability. Stronger than Madry's proof (did not have these restrictions).
We use $\text{DATALOG}^{c(\neq)}$ to find a tractable fragment for union of conjunctive queries with at most two inequalities

Theorem

Every 2-UCQ$\neq$ with:

- constant joins
- almost constant inequalities

can be expressed as a $\text{DATALOG}^{c(\neq)}$ program in data exchange.

Certain answers to this class of queries can be computed in polynomial time
We use $\text{DATALOG}^C(\neq)$ to find a tractable fragment for union of conjunctive queries with at most two inequalities.

**Theorem**

Every 2-UCQ $\neq$ with:

- constant joins
- almost constant inequalities

can be expressed as a $\text{DATALOG}^C(\neq)$ program in data exchange.

Certain answers to this class of queries can be computed in polynomial time.

- Removing any one of these conditions yields intractability
We use $\text{DATALOG}^C(\neq)$ to find a tractable fragment for union of conjunctive queries with at most two inequalities

**Theorem**

Every 2-UCQ $\neq$ with:
- constant joins
- almost constant inequalities

can be expressed as a $\text{DATALOG}^C(\neq)$ program in data exchange.

Certain answers to this class of queries can be computed in polynomial time

- Removing any one of this conditions yields to intractability
- Stronger than Madry’s proof (did not have these restrictions)
There is no hope for 3-CQ$\neq$

**Theorem**

There exists a query $Q$ in 3-CQ$\neq$ with
- constant joins
- almost constant inequalities

such that computing it’s certain answers is coNP-complete.
Outline

Formalization
- \textsc{Datalog}^C(\neq) \text{ programs}

Beyond union of conjunctive queries
- Expressive power of \textsc{Datalog}^C(\neq)
- New tractable classes of queries

Combined Complexity
- \textsc{Datalog}^C(\neq) \text{ and queries with inequalities}
- Restricting to \textsc{Lav} settings

Concluding remarks
Combined Complexity: a natural question

What is the complexity if we consider as inputs

- Database instance?
Combined Complexity: a natural question

What is the complexity if we consider as inputs

- Database instance
- Data exchange setting, query?
Combined Complexity: a natural question

What is the complexity if we consider as inputs
- Database instance
- Data exchange setting, query?

Kolaitis, Pantajja, Tan 06:
- Combined complexity of existence of solutions
- Lower bounds for query answering: 1-UCQ
Combined Complexity: a natural question

What is the complexity if we consider as inputs
- Database instance
- Data exchange setting, query?

Kolaitis, Pantajja, Tan 06:
- Combined complexity of existence of solutions
- Lower bounds for query answering: 1-UCQ

We study the combined complexity of query answering
- Tight lower bounds (single conjunctive queries)
- Results for DATALOG$^C(\neq)$ and related query languages
Combined Complexity for the general setting

**Theorem**

**Input:** Data exchange setting $\mathcal{M}$, query $Q$, instance $I$ and tuple $\bar{t}$

**Problem:** Is $\bar{t}$ in the certain answers of $Q$ for $I$ under $\mathcal{M}$?

**EXPTIME-complete** for $\text{DATALOG}^C(\neq)$ programs

▶ Same results hold for unions
▶ It follows from KPT06 that the problem is EXPTIME-complete for $1$-UCQ(\neq)$ programs
Combined Complexity for the general setting

**Theorem**

**Input:** Data exchange setting $\mathcal{M}$, query $Q$, instance $I$ and tuple $\bar{t}$

**Problem:** Is $\bar{t}$ in the certain answers of $Q$ for $I$ under $\mathcal{M}$?

- **EXPTIME-complete** for DATALOG$^{C(\neq)}$ programs
- **EXPTIME-complete** for 1-CQ$\neq$

▶ Same results hold for unions

It follows from KPT06 that the problem is **EXPTIME-complete** for 1-UCQ$\neq$
Combined Complexity for the general setting

Theorem

**Input:** Data exchange setting $\mathcal{M}$, query $Q$, instance $I$ and tuple $\bar{t}$

**Problem:** Is $\bar{t}$ in the certain answers of $Q$ for $I$ under $\mathcal{M}$?

- **$\text{EXPTIME}$-complete** for $\text{DATALOG}^{C(\neq)}$ programs
- **$\text{EXPTIME}$-complete** for $1$-$\text{CQ}^{\neq}$
- **$\text{coNEXPTIME}$-complete** for $k$-$\text{CQ}^{\neq}$, $k \geq 2$
- **$\text{coNEXPTIME}$-complete** for $\text{CQ}^{\neq}$
Combined Complexity for the general setting

**Theorem**

**Input:** Data exchange setting $\mathcal{M}$, query $Q$, instance $I$ and tuple $\bar{t}$

**Problem:** Is $\bar{t}$ in the certain answers of $Q$ for $I$ under $\mathcal{M}$?

- $\text{EXPTIME}$-complete for $\text{DATALOG}^c(\neq)$ programs
- $\text{EXPTIME}$-complete for $1$-$\text{CQ} \neq$
- $\text{coNEXPTIME}$-complete for $k$-$\text{CQ} \neq$, $k \geq 2$
- $\text{coNEXPTIME}$-complete for $\text{CQ} \neq$

- Same results hold for unions

- It follows from KPT06 that the problem is $\text{EXPTIME}$-complete for $1$-$\text{UCQ} \neq$
Lower combined complexity if we restrict to \textit{LAV} settings

A LAV setting is a data exchange setting where $\Sigma_{st}$ is of the form:

$$R(\bar{x}) \rightarrow \exists \bar{y} \psi(\bar{x}, \bar{y})$$

- Premises are single relational atoms
Lower combined complexity if we restrict to $\text{LAV}$ settings

A LAV setting is a data exchange settings where $\Sigma_{st}$ is of the form:

$$R(\bar{x}) \rightarrow \exists \bar{y} \psi(\bar{x}, \bar{y})$$

- Premises are single relational atoms

Very used in practice!
Lower combined complexity if we restrict to $\text{LAV}$ settings

A $\text{LAV}$ setting is a data exchange settings where $\sum_{st}$ is of the form:

$$R(\bar{x}) \rightarrow \exists \bar{y} \psi(\bar{x}, \bar{y})$$

- Premises are single relational atoms

Very used in practice!

Under $\text{LAV}$ settings, canonical universal solutions are of polynomial size (combined complexity)
Lower combined complexity if we restrict to \( LAV \) settings

**Theorem**

**Input:** \( LAV \) setting \( \mathcal{M} \), query \( Q \), instance \( I \) and tuple \( \bar{t} \)

**Problem:** Is \( \bar{t} \) in the certain answers of \( Q \) for \( I \) under \( \mathcal{M} \)?

\[ \text{EXPTIME-complete for } \text{Datalog}^C(\neq) \text{ programs} \]
Lower combined complexity if we restrict to \( LAV \) settings

**Theorem**

**Input:** \( LAV \) setting \( M \), query \( Q \), instance \( I \) and tuple \( \bar{t} \)

**Problem:** Is \( \bar{t} \) in the certain answers of \( Q \) for \( I \) under \( M \)?

- **Exptime-complete** for \( \text{Datalog}^C(\neq) \) programs
- **NP-complete** for 1-CQ\( \neq \)

▶ Same results hold for unions
Lower combined complexity if we restrict to LAV settings.

**Theorem**

**Input:** LAV setting $\mathcal{M}$, query $Q$, instance $I$ and tuple $\bar{t}$

**Problem:** Is $\bar{t}$ in the certain answers of $Q$ for $I$ under $\mathcal{M}$?

- **$\text{EXPTIME}$-complete** for $\text{Datalog}^C(\neq)$ programs
- **$\text{NP}$-complete** for 1-CQ$\neq$
- **$\Pi^p_2$-complete** for k-CQ$\neq$, $k \geq 2$
- **$\Pi^p_2$-complete** for CQ$\neq$
Lower combined complexity if we restrict to $LAV$ settings

**Theorem**

**Input:** $LAV$ setting $\mathcal{M}$, query $Q$, instance $I$ and tuple $\bar{t}$

**Problem:** Is $\bar{t}$ in the certain answers of $Q$ for $I$ under $\mathcal{M}$?

- $\text{EXPTIME}$-complete for $\text{DATALOG}^\text{C(\neq)}$ programs
- $\text{NP}$-complete for $1$-$\text{CQ} \neq$
- $\Pi^p_2$-complete for $k$-$\text{CQ} \neq$, $k \geq 2$
- $\Pi^p_2$-complete for $\text{CQ} \neq$

- Same results hold for unions
Outline

Formalization

- \( \text{Datalog}^C(\neq) \) programs

Beyond union of conjunctive queries

- Expressive power of \( \text{Datalog}^C(\neq) \)
- New tractable classes of queries

Combined Complexity

- \( \text{Datalog}^C(\neq) \) and queries with inequalities
- Restricting to \( \text{Lav} \) settings

Concluding remarks
We propose $\text{DATALOG}^C(\neq)$ as a query language for data exchange.

Study its properties:
- Preserved under homomorphisms
- Certain answers can be computed in polynomial time (data complexity)

$\text{DATALOG}^C(\neq)$, a tractable language that express negation:
- Union of conjunctive queries with one negated atom per disjunct
- A fragment of $2-\text{UCQ}^{\neq}$

We can use $\text{DATALOG}^C(\neq)$ to find tractable classes of queries.
Outline

Formalization
- $\text{DATALOG}^C(\neq)$ programs

Beyond union of conjunctive queries
- Expressive power of $\text{DATALOG}^C(\neq)$
- New tractable classes of queries

Combined Complexity
- $\text{DATALOG}^C(\neq)$ and queries with inequalities
- Restricting to LAV settings

Concluding remarks