Consistent Query Answering with Respect to Primary Keys

[Some] Past Research and Future Challenges

Jef Wijsen

University of Mons

Dagstuhl Seminar 24111, March 11–15, 2024

Table of Contents

Table of Contents

Inconsistent Data

Perrey Reeves

文A 18 lan

Article Talk Read Edit View history

From Wikipedia, the free encyclopedia

Perrey Reeves (born 1970 or 1971 (age 52–53))[1] is an American film and television actress. She is best known for her recurring role as Melissa Gold on the television series *Entourage* from 2004 to 2011 and Marissa Jones in the 2003 comedy *Old School*.

Perrey Reeves

Early life [edit]

Reeves was born in New York City and raised in New Hampshire,[2] the daughter of Dr. Alexander Reeves, a

Inconsistent Databases

ACTORS	<u>Name</u>	Gender	Age
	Jolie	F	48
	Pitt	M	59
	Pitt	M	60

Every actor has, at most, one gender and one age:

`ACTORS PRIMARY KEY(Name).`

Data cleaning takes time (and money). Can we already obtain "reliable" information by querying the inconsistent database?

Inconsistent Databases

ACTORS	Name	Gender	Age
	Jolie	F	48
	Pitt	M	59
	Pitt	M	60

Every actor has, at most, one gender and one age:

`ACTORS PRIMARY KEY(Name).`

Data cleaning takes time (and money). Can we already obtain "reliable" information by querying the inconsistent database?

Querying Inconsistent Databases

For ease of presentation, all queries return a Boolean (true/false).

ACTORS

<u>Name</u>	Gender	Age
Jolie	F	48
Pitt	M	59
Pitt	M	60

- Is Pitt's age 60?

$\exists y\,(\text{ACTORS}(\underline{\text{Pitt}}, y, 60))$ is "possibly false".

- Is Pitt older than Jolie?

$$\exists y \exists z \exists v \exists w \left(\begin{array}{l} \text{ACTORS}(\underline{\text{Pitt}}, y, z) \wedge \\ \text{ACTORS}(\underline{\text{Jolie}}, v, w) \wedge z > w \end{array} \right) \text{ is "certainly true".}$$

A block is a maximal set of tuples of the same relation that agree on their primary key (blocks are separated by dashed lines).
A repair (or possible world) is obtained by picking a single tuple from each block.

With this notion, "certainly true" means "true in every repair".

If 2 ages are stored for n actors, there are at least 2^n repairs.

Querying Inconsistent Databases

For ease of presentation, all queries return a Boolean (true/false).

ACTORS

<u>Name</u>	Gender	Age
Jolie	F	48
Pitt	M	59
Pitt	M	60

- Is Pitt's age 60?

$\exists y\,(\text{ACTORS}(\underline{\text{Pitt}}, y, 60))$ is "possibly false".

- Is Pitt older than Jolie?

$$\exists y \exists z \exists v \exists w \begin{pmatrix} \text{ACTORS}(\underline{\text{Pitt}}, y, z) \wedge \\ \text{ACTORS}(\underline{\text{Jolie}}, v, w) \wedge z > w \end{pmatrix} \text{ is “certainly true”}.$$

A block is a maximal set of tuples of the same relation that agree on their primary key (blocks are separated by dashed lines).
A repair (or possible world) is obtained by picking a single tuple from each block.

With this notion, "certainly true" means "true in every repair".

If 2 ages are stored for n actors, there are at least 2^n repairs.

Consistent Query Answering for Primary Keys

Given a Boolean query Q, define the following decision problem:

Problem CERTAINTY(Q)

Input: A database instance D that may violate primary-key constraints.

Question: Is Q true in every repair of D?

Example

If $Q_{60} = \exists y\,(\text{ACTORS}(\underline{\text{Pitt}}, y, 60))$, then the answer to CERTAINTY(Q_{60}) is "no" on our example database.

Remark

We assume that each relation name has a fixed primary key. Primary-key positions will be underlined. Primary keys can thus be derived from the query.

Consistent Query Answering for Primary Keys

Given a Boolean query Q, define the following decision problem:

Problem CERTAINTY(Q)

Input: A database instance D that may violate primary-key constraints.

Question: Is Q true in every repair of D?

Example

If $Q_{60} = \exists y\,(\mathrm{ACTORS}(\underline{\mathrm{Pitt}}, y, 60))$, then the answer to CERTAINTY(Q_{60}) is “no” on our example database.

Remark

We assume that each relation name has a fixed primary key. Primary-key positions will be underlined. Primary keys can thus be derived from the query.

Table of Contents

Solving CERTAINTY(Q)

Proposition

CERTAINTY(Q) *is in* coNP *for first-order queries* Q.

Proof.

A "'no" certificate is a repair that falsifies Q. $\square$

CERTAINTY(Q_{60}) is in FO, as the following are equivalent for every database instance D:

1. Q is true in every repair of D;
2. D satisfies $Q_{60} \wedge \neg\exists y \exists z\, (\text{ACTORS}(\underline{\text{Pitt}}, y, z) \wedge (z \neq 60))$.

Solving CERTAINTY(Q)

Proposition

CERTAINTY(Q) *is in* coNP *for first-order queries* Q.

Proof.

A "'no" certificate is a repair that falsifies Q. $\square$

CERTAINTY(Q_{60}) is in FO, as the following are equivalent for every database instance D:

1. Q is true in every repair of D;
2. D satisfies $Q_{60} \wedge \neg\exists y \exists z\,(\mathrm{ACTORS}(\underline{\mathrm{Pitt}}, y, z) \wedge (z \neq 60))$.

The Good, the Bad and the Ugly

Proposition

For $Q_{\text{good}} = \exists y\,(\text{ACTORS}(\underline{\text{Pitt}}, y, 60))$, *the decision problem* $\text{CERTAINTY}(Q_{\text{good}})$ *is in* FO.

Theorem ([W., 2010])

For $Q_{\text{bad}} = \exists x \exists y\,(R(\underline{x}, y) \wedge S(\underline{y}, x))$, *the decision problem* $\text{CERTAINTY}(Q_{\text{bad}})$ *is in* $\text{P} \setminus \text{FO}$ *(later, it was proven* L-*complete).*

Theorem ([Chomicki and Marcinkowski, 2005])

For $Q_{\text{ugly}} = \exists x_1 \exists x_2 \exists z\,(\text{ACTORS}(\underline{x_1}, \text{M}, z) \wedge \text{ACTORS}(\underline{x_2}, \text{F}, z))$, *the decision problem* $\text{CERTAINTY}(Q_{\text{ugly}})$ *is* coNP-*complete.*

The Good, the Bad and the Ugly

Proposition

For $Q_{\text{good}} = \exists y \, (\text{ACTORS}(\underline{\text{Pitt}}, y, 60))$, *the decision problem* CERTAINTY(Q_{good}) *is in* FO.

Theorem ([W., 2010])

For $Q_{\text{bad}} = \exists x \exists y \, \big(R(\underline{x}, y) \wedge S(\underline{y}, x)\big)$, *the decision problem* CERTAINTY(Q_{bad}) *is in* $\mathrm{P} \setminus \mathrm{FO}$ *(later, it was proven* L*-complete).*

Theorem ([Chomicki and Marcinkowski, 2005])

For $Q_{\text{ugly}} = \exists x_1 \exists x_2 \exists z \, (\text{ACTORS}(\underline{x_1}, \text{M}, z) \wedge \text{ACTORS}(\underline{x_2}, \text{F}, z))$, *the decision problem* CERTAINTY(Q_{ugly}) *is* coNP*-complete.*

The Good, the Bad and the Ugly

Proposition

For $Q_{\text{good}} = \exists y\,(\text{ACTORS}(\underline{\text{Pitt}}, y, 60))$*, the decision problem* $\text{CERTAINTY}(Q_{\text{good}})$ *is in* FO.

Theorem ([W., 2010])

For $Q_{\text{bad}} = \exists x \exists y\,\big(R(\underline{x}, y) \wedge S(\underline{y}, x)\big)$*, the decision problem* $\text{CERTAINTY}(Q_{\text{bad}})$ *is in* $\text{P} \setminus \text{FO}$ *(later, it was proven L-complete).*

Theorem ([Chomicki and Marcinkowski, 2005])

For $Q_{\text{ugly}} = \exists x_1 \exists x_2 \exists z\,(\text{ACTORS}(\underline{x_1}, \text{M}, z) \wedge \text{ACTORS}(\underline{x_2}, \text{F}, z))$*, the decision problem* $\text{CERTAINTY}(Q_{\text{ugly}})$ *is coNP-complete.*

The Good, the Bad and the Ugly

Proposition

For $Q_{\text{good}} = \exists y\,(\text{ACTORS}(\underline{\text{Pitt}}, y, 60))$, *the decision problem* CERTAINTY(Q_{good}) *is in* FO.

Theorem ([W., 2010])

For $Q_{\text{bad}} = \exists x \exists y\,\big(R(\underline{x}, y) \wedge S(\underline{y}, x)\big)$, *the decision problem* CERTAINTY(Q_{bad}) *is in* P $\setminus$ FO *(later, it was proven* L*-complete).*

Theorem ([Chomicki and Marcinkowski, 2005])

For $Q_{\text{ugly}} = \exists x_1 \exists x_2 \exists z\,\big(\text{ACTORS}(\underline{x_1}, \text{M}, z) \wedge \text{ACTORS}(\underline{x_2}, \text{F}, z)\big)$, *the decision problem* CERTAINTY(Q_{ugly}) *is* coNP*-complete.*

Research Agenda

- We aim to go beyond the task of determining CERTAINTY(Q) for individual queries Q.
- For "reasonable" classes $\mathcal{C}$ of queries, write an algorithm for the following problem:

Complexity Classification Task

Input: A query Q in the class $\mathcal{C}$.

Task: The computational complexity of CERTAINTY(Q), in terms of complexity classes like FO, P, coNP-complete,...

Research Agenda

- We aim to go beyond the task of determining CERTAINTY(Q) for individual queries Q.
- For "reasonable" classes $\mathcal{C}$ of queries, write an algorithm for the following problem:

Complexity Classification Task

Input: A query Q in the class $\mathcal{C}$.

Task: The computational complexity of CERTAINTY(Q), in terms of complexity classes like FO, P, coNP-complete,. . .

Which Query Classes Are "Reasonable"?

- The class of (Boolean) conjunctive queries (a.k.a. Select-Project-Join queries):

$$\exists \vec{u} \left(R_1(\underline{\vec{x}_1}, \vec{y}_1) \wedge R_2(\underline{\vec{x}_2}, \vec{y}_2) \wedge \cdots \wedge R_n(\underline{\vec{x}_n}, \vec{y}_n) \right). \qquad (1)$$

- The class of disjunctions of conjunctive queries (a.k.a. UCQ queries):

$$Q_1 \vee Q_2 \vee \cdots \vee Q_m,$$

where each Q_i is of the form (1).

Which Query Classes Are "Reasonable"?

- The class of (Boolean) conjunctive queries (a.k.a. Select-Project-Join queries):

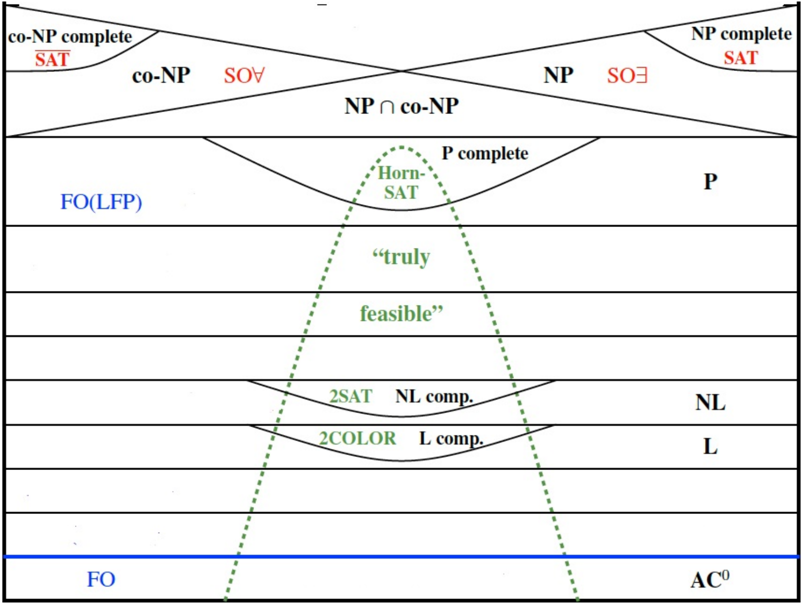
$$\exists \vec{u} \left(R_1(\underline{\vec{x}_1}, \vec{y}_1) \wedge R_2(\underline{\vec{x}_2}, \vec{y}_2) \wedge \cdots \wedge R_n(\underline{\vec{x}_n}, \vec{y}_n) \right). \qquad (1)$$

- The class of disjunctions of conjunctive queries (a.k.a. UCQ queries):

$$Q_1 \vee Q_2 \vee \cdots \vee Q_m,$$

  where each Q_i is of the form (1).

Which Complexity Classes?



Classifying CERTAINTY(Q) in P/coNP-complete is Hard

Conjecture

If Q is a disjunction of conjunctive queries, then CERTAINTY(Q) *is in* P *or* coNP*-complete.*

Theorem ([Fontaine, 2015])

The above conjecture implies Bulatov's dichotomy theorem for the conservative constraint satisfaction problem (CSP).

Classifying CERTAINTY(Q) in P/coNP-complete is Hard

Conjecture

If Q is a disjunction of conjunctive queries, then CERTAINTY(Q) *is in* P *or* coNP*-complete.*

Theorem ([Fontaine, 2015])

The above conjecture implies Bulatov's dichotomy theorem for the conservative constraint satisfaction problem (CSP).

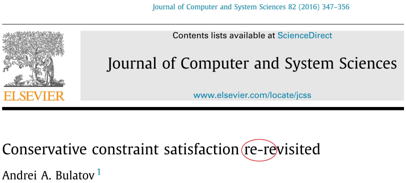
Classifying CERTAINTY(Q) in P/coNP-complete is Hard

Conjecture

If Q is a disjunction of conjunctive queries, then CERTAINTY(Q) *is in* P *or* coNP*-complete.*

Theorem ([Fontaine, 2015])

The above conjecture implies Bulatov's dichotomy theorem for the conservative constraint satisfaction problem (CSP).


Journal of Computer and System Sciences 82 (2016) 347–356

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

ELSEVIER

Conservative constraint satisfaction re-revisited

Andrei A. Bulatov [1]

Is it Easier for Conjunctive Queries?

Conjecture

If Q is of the form $\exists \vec{u} \left(R_1(\underline{\vec{x}_1}, \vec{y}_1) \wedge \cdots \wedge R_n(\underline{\vec{x}_n}, \vec{y}_n) \right)$, then CERTAINTY($Q$) *is in* P *or* coNP*-complete.*

Theorem ([Koutris and W., 2017])

The above conjecture holds under the assumption that $R_i \neq R_j$ whenever $i \neq j$.

Somewhat later, if was proven that for every self-join-free CQ Q, CERTAINTY(Q) is either in FO, L-complete, or coNP-complete.

Theorem ([Padmanabha et al., 2023])

The above conjecture holds under the assumption that $n = 2$.

Theorem ([Koutris et al., 2021])

The above conjecture holds for queries of the form $\exists x_1 \cdots \exists x_{n+1} \left(R_1(\underline{x_1}, x_2) \wedge R_2(\underline{x_2}, x_3) \wedge \cdots \wedge R_n(\underline{x_n}, x_{n+1}) \right)$.

Is it Easier for Conjunctive Queries?

Conjecture

If Q is of the form $\exists\vec{u}\left(R_1(\underline{\vec{x}_1},\vec{y}_1)\wedge\cdots\wedge R_n(\underline{\vec{x}_n},\vec{y}_n)\right)$, then CERTAINTY($Q$) *is in* P *or* coNP*-complete.*

Theorem ([Koutris and W., 2017])

The above conjecture holds under the assumption that $R_i \neq R_j$ whenever $i \neq j$.

Somewhat later, if was proven that for every self-join-free CQ Q, CERTAINTY(Q) is either in FO, L-complete, or coNP-complete.

Theorem ([Padmanabha et al., 2023])

The above conjecture holds under the assumption that $n = 2$.

Theorem ([Koutris et al., 2021])

The above conjecture holds for queries of the form $\exists x_1\cdots\exists x_{n+1}\left(R_1(\underline{x_1},x_2)\wedge R_2(\underline{x_2},x_3)\wedge\cdots\wedge R_n(\underline{x_n},x_{n+1})\right)$.

Is it Easier for Conjunctive Queries?

Conjecture

If Q is of the form $\exists\vec{u}\left(R_1(\underline{\vec{x}_1},\vec{y}_1)\land\cdots\land R_n(\underline{\vec{x}_n},\vec{y}_n)\right)$, *then* CERTAINTY($Q$) *is in* P *or* coNP*-complete.*

Theorem ([Koutris and W., 2017])

The above conjecture holds under the assumption that $R_i \neq R_j$ whenever $i \neq j$.

Somewhat later, if was proven that for every self-join-free CQ Q, CERTAINTY(Q) is either in FO, L-complete, or coNP-complete.

Theorem ([Padmanabha et al., 2023])

The above conjecture holds under the assumption that $n = 2$.

Theorem ([Koutris et al., 2021])

The above conjecture holds for queries of the form $\exists x_1\cdots\exists x_{n+1}\left(R_1(\underline{x_1},x_2)\land R_2(\underline{x_2},x_3)\land\cdots\land R_n(\underline{x_n},x_{n+1})\right)$.

Is it Easier for Conjunctive Queries?

Conjecture

If Q is of the form $\exists\vec{u}\left(R_1(\underline{\vec{x}_1},\vec{y}_1)\wedge\cdots\wedge R_n(\underline{\vec{x}_n},\vec{y}_n)\right)$*, then* CERTAINTY($Q$) *is in* P *or* coNP*-complete.*

Theorem ([Koutris and W., 2017])

The above conjecture holds under the assumption that $R_i \neq R_j$ whenever $i \neq j$.

Somewhat later, if was proven that for every self-join-free CQ Q, CERTAINTY(Q) is either in FO, L-complete, or coNP-complete.

Theorem ([Padmanabha et al., 2023])

The above conjecture holds under the assumption that $n = 2$.

Theorem ([Koutris et al., 2021])

The above conjecture holds for queries of the form $\exists x_1\cdots\exists x_{n+1}\left(R_1(\underline{x_1},x_2)\wedge R_2(\underline{x_2},x_3)\wedge\cdots\wedge R_n(\underline{x_n},x_{n+1})\right)$.

Is it Easier for Conjunctive Queries?

Conjecture

If Q is of the form $\exists\vec{u}\left(R_1(\underline{\vec{x}_1}, \vec{y}_1) \wedge \cdots \wedge R_n(\underline{\vec{x}_n}, \vec{y}_n)\right)$, then CERTAINTY($Q$) *is in* P *or* coNP*-complete.*

Theorem ([Koutris and W., 2017])

The above conjecture holds under the assumption that $R_i \neq R_j$ whenever $i \neq j$.

Somewhat later, if was proven that for every self-join-free CQ Q, CERTAINTY(Q) is either in FO, L-complete, or coNP-complete.

Theorem ([Padmanabha et al., 2023])

The above conjecture holds under the assumption that $n = 2$.

Theorem ([Koutris et al., 2021])

The above conjecture holds for queries of the form
$\exists x_1 \cdots \exists x_{n+1}\left(R_1(\underline{x_1}, x_2) \wedge R_2(\underline{x_2}, x_3) \wedge \cdots \wedge R_n(\underline{x_n}, x_{n+1})\right)$.

Table of Contents

The Good Among the Good, the Bad and the Ugly

A directed graph, called attack graph, is defined for every conjunctive query.

Theorem ([Koutris and W., 2017])

Let $Q = \exists \vec{u} \left(R_1(\underline{\vec{x}_1}, \vec{y}_1) \land \cdots \land R_n(\underline{\vec{x}_n}, \vec{y}_n) \right)$ *with* $R_i \neq R_j$ *for* $i \neq j$. *Then,*

- *if Q's attack graph is acyclic, then* CERTAINTY(Q) *is in* FO;
- *if Q's attack graph is cyclic, then* CERTAINTY(Q) *is* L*-hard.*

The Good Among the Good, the Bad and the Ugly

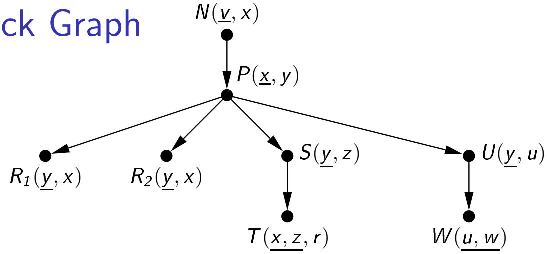
A directed graph, called attack graph, is defined for every conjunctive query.

Theorem ([Koutris and W., 2017])

Let $Q = \exists \vec{u} \left(R_1(\underline{\vec{x}_1}, \vec{y}_1) \wedge \cdots \wedge R_n(\underline{\vec{x}_n}, \vec{y}_n) \right)$ *with* $R_i \neq R_j$ *for* $i \neq j$*. Then,*

- *if Q's attack graph is acyclic, then* CERTAINTY(Q) *is in* FO*;*
- *if Q's attack graph is cyclic, then* CERTAINTY(Q) *is* L*-hard.*

Attack Graph



$N^+ = \{v\}$

$P^+ = \{x\}$

${R_1}^+ = \{y, x, z, r, u\}$

${R_2}^+ = \{y, x, z, r, u\}$

$S^+ = \{y, x, u\}$

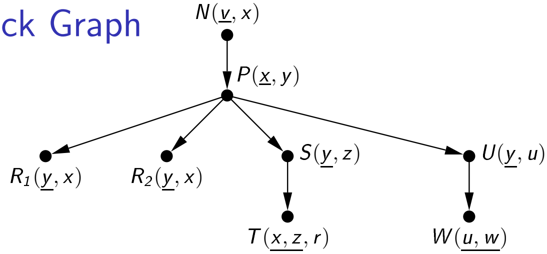
$U^+ = \{y, x, z, r\}$

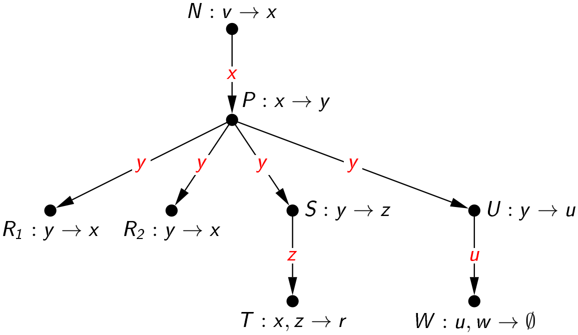
$T^+ = \{x, z, y, u\}$

$W^+ = \{u, w\}$

S^+, e.g., is the closure of S's key w.r.t. all other FDs.
S can attack with $z \notin S^+$.

Attack Graph



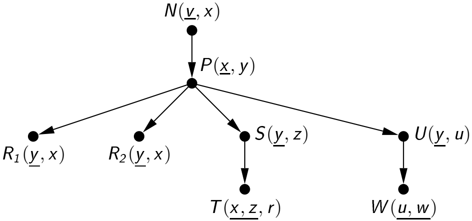


$$N^+ = \{v\}$$
$$P^+ = \{x\}$$
$$R_1{}^+ = \{y, x, z, r, u\}$$
$$R_2{}^+ = \{y, x, z, r, u\}$$
$$S^+ = \{y, x, u\}$$
$$U^+ = \{y, x, z, r\}$$
$$T^+ = \{x, z, y, u\}$$
$$W^+ = \{u, w\}$$

S^+, e.g., is the closure of S's key w.r.t. all other FDs.
S can attack with $z \notin S^+$.

Attack Graph and (Consistent) First-Order Rewriting

Q :



We construct a first-order formula φ_N such that for every database:

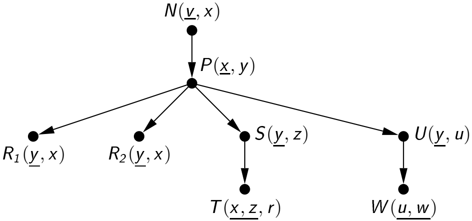
φ_N is true in the database $\iff$ Q is true in every repair.

$$\varphi_N := \exists v \left(\exists x \left(N(\underline{v}, x) \right) \land \neg \exists x \left(N(\underline{v}, x) \land \neg \varphi_P(v, x) \right) \right),$$

where $\varphi_P(v, x)$ is a rewriting of the conjunctive query whose atoms are the atoms of Q except $N(\underline{v}, x)$, in which variables v and x are free.
The empty query rewrites to `true`.

Attack Graph and (Consistent) First-Order Rewriting

Q :



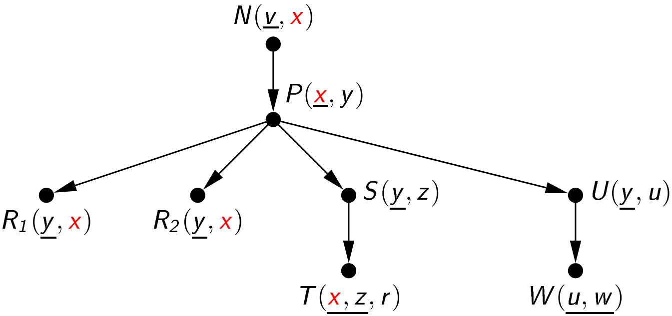
We construct a first-order formula φ_N such that for every database:

φ_N is true in the database $\iff$ Q is true in every repair.

$$\varphi_N := \exists v \left(\exists x \left(N(\underline{v}, x)\right) \wedge \neg \exists x \left(N(\underline{v}, x) \wedge \neg \varphi_P(v, x)\right)\right),$$

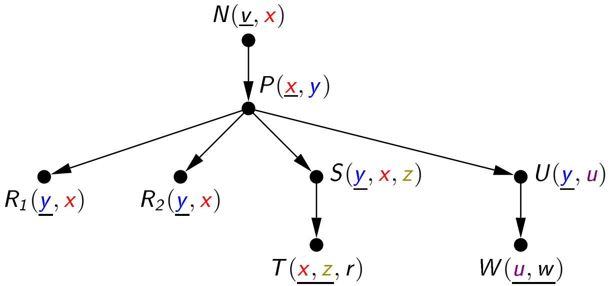
where $\varphi_P(v, x)$ is a rewriting of the conjunctive query whose atoms are the atoms of Q except $N(\underline{v}, x)$, in which variables v and x are free.
The empty query rewrites to `true`.

Attack Graph $\neq$ Join Tree



The subgraph induced by atoms that contain x is not connected.

Attack Graph that Is a Join Tree

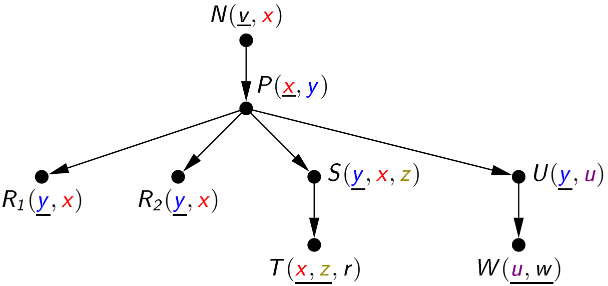


Moreover, every internal node V has zero indegree in the attack graph of the subquery rooted at V ($V \in \{P, S, U\}$).
Such a join tree is called a Pair-Pruning Join Tree (PPJT).
Yannakakis' algorithm extends to the inconsistent setting:

Theorem ([Fan et al., 2023])

If Q has a PPJT, then CERTAINTY(Q) *is in* LIN *(i.e., problems solvable in linear time).*

Attack Graph that Is a Join Tree

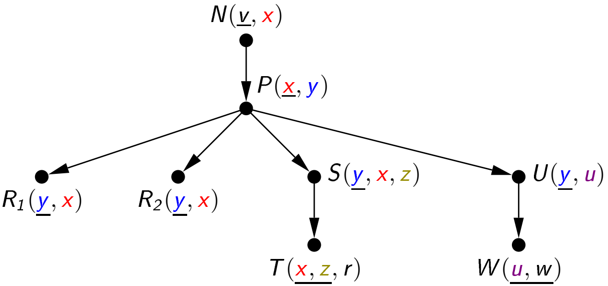


Moreover, every internal node V has zero indegree in the attack graph of the subquery rooted at V ($V \in \{P, S, U\}$).
Such a join tree is called a Pair-Pruning Join Tree (PPJT).
Yannakakis' algorithm extends to the inconsistent setting:

Theorem ([Fan et al., 2023])

If Q has a PPJT, then CERTAINTY(Q) *is in* LIN *(i.e., problems solvable in linear time).*

Attack Graph that Is a Join Tree

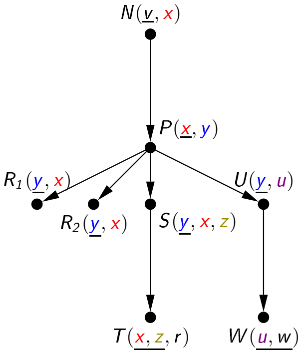


Moreover, every internal node V has zero indegree in the attack graph of the subquery rooted at V ($V \in \{P, S, U\}$).
Such a join tree is called a Pair-Pruning Join Tree (PPJT).
Yannakakis' algorithm extends to the inconsistent setting:

Theorem ([Fan et al., 2023])

If Q has a PPJT, then CERTAINTY(Q) *is in* LIN *(i.e., problems solvable in linear time).*

Yannakakis+Pruning



$$T^{\text{join}}(x, z) \leftarrow T(\underline{x, z}, r)$$
$$W^{\text{join}}(u) \leftarrow W(\underline{u, w})$$

$$\mathit{Answer}(\text{yes}) \leftarrow N(\underline{v}, x) \land \neg N^{\text{fadingkey}}(v)$$
$$N^{\text{fadingkey}}(v) \leftarrow N(\underline{v}, x) \land \neg P^{\text{join}}(x)$$

$$P^{\text{join}}(x) \leftarrow P(\underline{x}, y) \land \neg P^{\text{fadingkey}}(x)$$
$$P^{\text{fadingkey}}(x) \leftarrow P(\underline{x}, y) \land \neg U^{\text{join}}(y)$$
$$P^{\text{fadingkey}}(x) \leftarrow P(\underline{x}, y) \land \neg S^{\text{join}}(x, y)$$
$$P^{\text{fadingkey}}(x) \leftarrow P(\underline{x}, y) \land \neg R_i^{\text{join}}(x, y)$$

$$U^{\text{join}}(y) \leftarrow U(\underline{y}, u) \land \neg U^{\text{fadingkey}}(y)$$
$$U^{\text{fadingkey}}(y) \leftarrow U(\underline{y}, u) \land \neg W^{\text{join}}(u)$$

$$S^{\text{join}}(x, y) \leftarrow S(\underline{y}, x, z) \land \neg S^{\text{fadingkey}}(y)$$
$$S^{\text{fadingkey}}(y) \leftarrow S(\underline{y}, x, z) \land S(\underline{y}, x', z) \land x \neq x'$$
$$S^{\text{fadingkey}}(y) \leftarrow S(\underline{y}, x, z) \land \neg T^{\text{join}}(x, z)$$

$$R_i^{\text{join}}(x, y) \leftarrow R_i(\underline{y}, x) \land \neg R_i^{\text{fadingkey}}(y)$$
$$R_i^{\text{fadingkey}}(y) \leftarrow R_i(\underline{y}, x) \land R_i(\underline{y}, x') \land x \neq x'$$
$$(1 \leq i \leq 2)$$

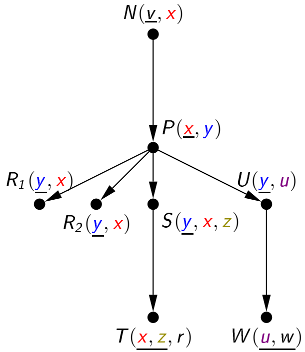
Yannakakis+Pruning

$$Answer(\text{yes}) \leftarrow N(\underline{v}, x) \land \neg N^{\text{fadingkey}}(v)$$
$$N^{\text{fadingkey}}(v) \leftarrow N(\underline{v}, x) \land \neg P^{\text{join}}(x)$$

$$P^{\text{join}}(x) \leftarrow P(\underline{x}, y) \land \neg P^{\text{fadingkey}}(x)$$
$$P^{\text{fadingkey}}(x) \leftarrow P(\underline{x}, y) \land \neg U^{\text{join}}(y)$$
$$P^{\text{fadingkey}}(x) \leftarrow P(\underline{x}, y) \land \neg S^{\text{join}}(x, y)$$
$$P^{\text{fadingkey}}(x) \leftarrow P(\underline{x}, y) \land \neg R_i^{\text{join}}(x, y)$$

$$U^{\text{join}}(y) \leftarrow U(\underline{y}, u) \land \neg U^{\text{fadingkey}}(y)$$
$$U^{\text{fadingkey}}(y) \leftarrow U(\underline{y}, u) \land \neg W^{\text{join}}(u)$$

$$S^{\text{join}}(x, y) \leftarrow S(\underline{y}, x, z) \land \neg S^{\text{fadingkey}}(y)$$
$$S^{\text{fadingkey}}(y) \leftarrow S(\underline{y}, x, z) \land S(\underline{y}, x', z) \land x \neq x'$$
$$S^{\text{fadingkey}}(y) \leftarrow S(\underline{y}, x, z) \land \neg T^{\text{join}}(x, z)$$



$$T^{\text{join}}(x, z) \leftarrow T(\underline{x, z}, r)$$
$$W^{\text{join}}(u) \leftarrow W(\underline{u, w})$$

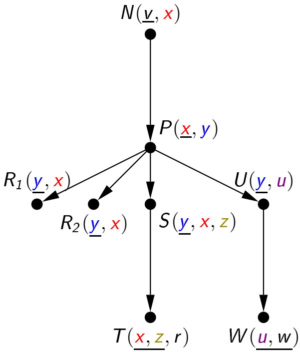
$$R_i^{\text{join}}(x, y) \leftarrow R_i(\underline{y}, x) \land \neg R_i^{\text{fadingkey}}(y)$$
$$R_i^{\text{fadingkey}}(y) \leftarrow R_i(\underline{y}, x) \land R_i(\underline{y}, x') \land x \neq x'$$
$$(1 \leq i \leq 2)$$

Yannakakis+Pruning

$$Answer(\text{yes}) \leftarrow N(\underline{v}, x) \wedge \neg N^{\text{fadingkey}}(v)$$
$$N^{\text{fadingkey}}(v) \leftarrow N(\underline{v}, x) \wedge \neg P^{\text{join}}(x)$$

$$P^{\text{join}}(x) \leftarrow P(\underline{x}, y) \wedge \neg P^{\text{fadingkey}}(x)$$
$$P^{\text{fadingkey}}(x) \leftarrow P(\underline{x}, y) \wedge \neg U^{\text{join}}(y)$$
$$P^{\text{fadingkey}}(x) \leftarrow P(\underline{x}, y) \wedge \neg S^{\text{join}}(x, y)$$
$$P^{\text{fadingkey}}(x) \leftarrow P(\underline{x}, y) \wedge \neg R_i^{\text{join}}(x, y)$$

$$U^{\text{join}}(y) \leftarrow U(\underline{y}, u) \wedge \neg U^{\text{fadingkey}}(y)$$
$$U^{\text{fadingkey}}(y) \leftarrow U(\underline{y}, u) \wedge \neg W^{\text{join}}(u)$$



$$S^{\text{join}}(x, y) \leftarrow S(\underline{y}, x, z) \wedge \neg S^{\text{fadingkey}}(y)$$
$$S^{\text{fadingkey}}(y) \leftarrow S(\underline{y}, x, z) \wedge S(\underline{y}, x', z) \wedge x \neq x'$$
$$S^{\text{fadingkey}}(y) \leftarrow S(\underline{y}, x, z) \wedge \neg T^{\text{join}}(x, z)$$

$$T^{\text{join}}(x, z) \leftarrow T(\underline{x, z}, r)$$
$$W^{\text{join}}(u) \leftarrow W(\underline{u, w})$$

$$R_i^{\text{join}}(x, y) \leftarrow R_i(\underline{y}, x) \wedge \neg R_i^{\text{fadingkey}}(y)$$
$$R_i^{\text{fadingkey}}(y) \leftarrow R_i(\underline{y}, x) \wedge R_i(\underline{y}, x') \wedge x \neq x'$$
$$(1 \leq i \leq 2)$$

Yannakakis+Pruning

$$\mathit{Answer}(\text{yes}) \leftarrow N(\underline{v}, x) \wedge \neg N^{\text{fadingkey}}(v)$$

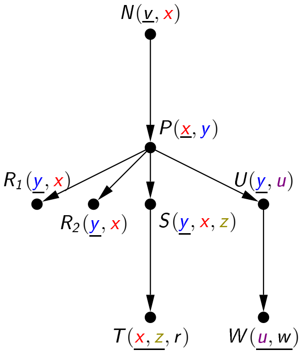
$$N^{\text{fadingkey}}(v) \leftarrow N(\underline{v}, x) \wedge \neg P^{\text{join}}(x)$$

$$P^{\text{join}}(x) \leftarrow P(\underline{x}, y) \wedge \neg P^{\text{fadingkey}}(x)$$

$$P^{\text{fadingkey}}(x) \leftarrow P(\underline{x}, y) \wedge \neg U^{\text{join}}(y)$$

$$P^{\text{fadingkey}}(x) \leftarrow P(\underline{x}, y) \wedge \neg S^{\text{join}}(x, y)$$

$$P^{\text{fadingkey}}(x) \leftarrow P(\underline{x}, y) \wedge \neg R_i^{\text{join}}(x, y)$$



$$U^{\text{join}}(y) \leftarrow U(\underline{y}, u) \wedge \neg U^{\text{fadingkey}}(y)$$

$$U^{\text{fadingkey}}(y) \leftarrow U(\underline{y}, u) \wedge \neg W^{\text{join}}(u)$$

$$S^{\text{join}}(x, y) \leftarrow S(\underline{y}, x, z) \wedge \neg S^{\text{fadingkey}}(y)$$

$$S^{\text{fadingkey}}(y) \leftarrow S(\underline{y}, x, z) \wedge S(\underline{y}, x', z) \wedge x \neq x'$$

$$S^{\text{fadingkey}}(y) \leftarrow S(\underline{y}, x, z) \wedge \neg T^{\text{join}}(x, z)$$

$$T^{\text{join}}(x, z) \leftarrow T(\underline{x, z}, r)$$

$$W^{\text{join}}(u) \leftarrow W(\underline{u, w})$$

$$R_i^{\text{join}}(x, y) \leftarrow R_i(\underline{y}, x) \wedge \neg R_i^{\text{fadingkey}}(y)$$

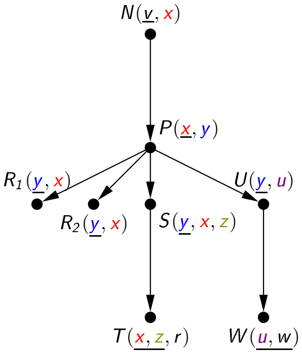
$$R_i^{\text{fadingkey}}(y) \leftarrow R_i(\underline{y}, x) \wedge R_i(\underline{y}, x') \wedge x \neq x'$$

$$(1 \leq i \leq 2)$$

Yannakakis+Pruning

$$Answer(\text{yes}) \leftarrow N(\underline{v}, x) \land \neg N^{\text{fadingkey}}(v)$$

$$N^{\text{fadingkey}}(v) \leftarrow N(\underline{v}, x) \land \neg P^{\text{join}}(x)$$



$$T^{\text{join}}(x, z) \leftarrow T(\underline{x, z}, r)$$

$$W^{\text{join}}(u) \leftarrow W(\underline{u, w})$$

$$P^{\text{join}}(x) \leftarrow P(\underline{x}, y) \land \neg P^{\text{fadingkey}}(x)$$

$$P^{\text{fadingkey}}(x) \leftarrow P(\underline{x}, y) \land \neg U^{\text{join}}(y)$$

$$P^{\text{fadingkey}}(x) \leftarrow P(\underline{x}, y) \land \neg S^{\text{join}}(x, y)$$

$$P^{\text{fadingkey}}(x) \leftarrow P(\underline{x}, y) \land \neg R_i^{\text{join}}(x, y)$$

$$U^{\text{join}}(y) \leftarrow U(\underline{y}, u) \land \neg U^{\text{fadingkey}}(y)$$

$$U^{\text{fadingkey}}(y) \leftarrow U(\underline{y}, u) \land \neg W^{\text{join}}(u)$$

$$S^{\text{join}}(x, y) \leftarrow S(\underline{y}, x, z) \land \neg S^{\text{fadingkey}}(y)$$

$$S^{\text{fadingkey}}(y) \leftarrow S(\underline{y}, x, z) \land S(\underline{y}, x', z) \land x \neq x'$$

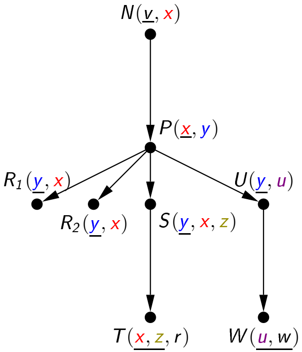
$$S^{\text{fadingkey}}(y) \leftarrow S(\underline{y}, x, z) \land \neg T^{\text{join}}(x, z)$$

$$R_i^{\text{join}}(x, y) \leftarrow R_i(\underline{y}, x) \land \neg R_i^{\text{fadingkey}}(y)$$

$$R_i^{\text{fadingkey}}(y) \leftarrow R_i(\underline{y}, x) \land R_i(\underline{y}, x') \land x \neq x'$$

$$(1 \leq i \leq 2)$$

Yannakakis+Pruning



$$T^{\text{join}}(x, z) \leftarrow T(\underline{x, z}, r)$$

$$W^{\text{join}}(u) \leftarrow W(\underline{u, w})$$

$$\mathit{Answer}(\text{yes}) \leftarrow N(\underline{v}, x) \wedge \neg N^{\text{fadingkey}}(v)$$

$$N^{\text{fadingkey}}(v) \leftarrow N(\underline{v}, x) \wedge \neg P^{\text{join}}(x)$$

$$P^{\text{join}}(x) \leftarrow P(\underline{x}, y) \wedge \neg P^{\text{fadingkey}}(x)$$

$$P^{\text{fadingkey}}(x) \leftarrow P(\underline{x}, y) \wedge \neg U^{\text{join}}(y)$$

$$P^{\text{fadingkey}}(x) \leftarrow P(\underline{x}, y) \wedge \neg S^{\text{join}}(x, y)$$

$$P^{\text{fadingkey}}(x) \leftarrow P(\underline{x}, y) \wedge \neg R_i^{\text{join}}(x, y)$$

$$U^{\text{join}}(y) \leftarrow U(\underline{y}, u) \wedge \neg U^{\text{fadingkey}}(y)$$

$$U^{\text{fadingkey}}(y) \leftarrow U(\underline{y}, u) \wedge \neg W^{\text{join}}(u)$$

$$S^{\text{join}}(x, y) \leftarrow S(\underline{y}, x, z) \wedge \neg S^{\text{fadingkey}}(y)$$

$$S^{\text{fadingkey}}(y) \leftarrow S(\underline{y}, x, z) \wedge S(\underline{y}, x', z) \wedge x \neq x'$$

$$S^{\text{fadingkey}}(y) \leftarrow S(\underline{y}, x, z) \wedge \neg T^{\text{join}}(x, z)$$

$$R_i^{\text{join}}(x, y) \leftarrow R_i(\underline{y}, x) \wedge \neg R_i^{\text{fadingkey}}(y)$$

$$R_i^{\text{fadingkey}}(y) \leftarrow R_i(\underline{y}, x) \wedge R_i(\underline{y}, x') \wedge x \neq x'$$

$$(1 \leq i \leq 2)$$

Observation Regarding Correctness

R_i-blocks of size ≥ 2 can be ignored. For example,

$$R_1 \begin{array}{|cc} \underline{y} & x \\ \hline a & c_1 \\ a & c_2 \end{array} \qquad R_2 \begin{array}{|cc} \underline{y} & x \\ \hline a & c_1 \\ a & c_2 \end{array}$$

To construct a repair that falsifies the query, pick $R_1(\underline{a}, c_i)$ and $R_2(\underline{a}, c_j)$ such that $c_i \neq c_j$.

LinCQA

- LinCQA is a system that takes as input any query with a PPJT and outputs rewritings in both SQL and non-recursive Datalog with negation.
- `https://github.com/xiatingouyang/LinCQA/`
- See [Fan et al., 2023] for experiments.

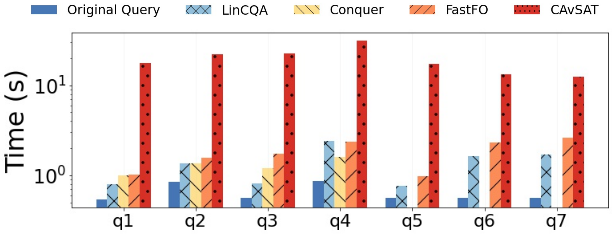


Table of Contents

Table of Contents

Range Consistent Query Answering [Arenas et al., 2001]

For queries returning numbers instead of Booleans.

For ease of presentation, all queries return a single number.

ACTORS

Name	Gender	Age
Jolie	F	48
Pitt	F	59
Pitt	M	60
Reeves	F	52
Reeves	F	53

Get the age of the oldest actress:

$$\mathtt{MAX}(z) \leftarrow \mathsf{ACTORS}(\underline{x}, \mathsf{F}, z).$$

- The lowest answer across all repairs is $\mathtt{MAX}(\{48, 52\}) = 52$;
- the greatest answer across all repairs is $\mathtt{MAX}(\{48, 59, 53\}) = 59$;
- the interval $[52, 59]$ is called the range consistent answer.

Range Consistent Query Answering [Arenas et al., 2001]

For queries returning numbers instead of Booleans.

For ease of presentation, all queries return a single number.

ACTORS

<u>Name</u>	Gender	Age
Jolie	F	48
Pitt	F	59
Pitt	M	60
Reeves	F	52
Reeves	F	53

Get the age of the oldest actress:

$$\mathtt{MAX}(z) \leftarrow \mathsf{ACTORS}(\underline{x}, \mathsf{F}, z).$$

- The lowest answer across all repairs is $\mathtt{MAX}(\{48, 52\}) = 52$;
- the greatest answer across all repairs is $\mathtt{MAX}(\{48, 59, 53\}) = 59$;
- the interval $[52, 59]$ is called the range consistent answer.

Range Consistent Query Answering [Arenas et al., 2001]

For queries returning numbers instead of Booleans.

For ease of presentation, all queries return a single number.

ACTORS

<u>Name</u>	Gender	Age
Jolie	F	48
Pitt	F	59
Pitt	M	60
Reeves	F	52
Reeves	F	53

Get the age of the oldest actress:

$$\mathtt{MAX}(z) \leftarrow \mathsf{ACTORS}(\underline{x}, \mathsf{F}, z).$$

- The lowest answer across all repairs is $\mathtt{MAX}(\{48, 52\}) = 52$;
- the greatest answer across all repairs is $\mathtt{MAX}(\{48, 59, 53\}) = 59$;
- the interval $[52, 59]$ is called the range consistent answer.

Formal Setting

- Numerical terms $f()$ expressible in the (safe) rule format

$$\texttt{AGG}(r) \leftarrow R_1(\underline{\vec{x}_1}, \vec{y}_1) \wedge R_2(\underline{\vec{x}_2}, \vec{y}_2) \wedge \cdots \wedge R_n(\underline{\vec{x}_n}, \vec{y}_n), \qquad (2)$$

  where r is either a numerical variable or a constant, and AGG is an aggregate operator (e.g., MAX, MIN, SUM, COUNT, AVG); assume $R_i \neq R_j$ if $i \neq j$.
- Given a database instance, let $f^+()$ and $f^-()$ be, respectively, the greatest and smallest values of $f()$ across all repairs.
- Aggregate logic $\mathcal{L}_{\mathrm{aggr}}$ [Hella et al., 2001]: FOL + aggregation.
- Question in [Fuxman, 2007] and [Dixit and Kolaitis, 2022]:

  When can $f^+()$ and $f^-()$ be expressed in $\mathcal{L}_{\mathrm{aggr}}$?

  1. $f^+()$ and $f^-()$ are not expressible in $\mathcal{L}_{\mathrm{aggr}}$ if the attack graph of (2) is cyclic (because queries in $\mathcal{L}_{\mathrm{aggr}}$ are Hanf-local).
  2. Does the converse hold?

Formal Setting

- Numerical terms $f()$ expressible in the (safe) rule format

$$\texttt{AGG}(r) \leftarrow R_1(\underline{\vec{x_1}}, \vec{y_1}) \wedge R_2(\underline{\vec{x_2}}, \vec{y_2}) \wedge \cdots \wedge R_n(\underline{\vec{x_n}}, \vec{y_n}), \quad (2)$$

  where r is either a numerical variable or a constant, and AGG is an aggregate operator (e.g., MAX, MIN, SUM, COUNT, AVG); assume $R_i \neq R_j$ if $i \neq j$.
- Given a database instance, let $f^+()$ and $f^-()$ be, respectively, the greatest and smallest values of $f()$ across all repairs.
- Aggregate logic $\mathcal{L}_{\mathrm{aggr}}$ [Hella et al., 2001]: FOL + aggregation.
- Question in [Fuxman, 2007] and [Dixit and Kolaitis, 2022]:

  When can $f^+()$ and $f^-()$ be expressed in $\mathcal{L}_{\mathrm{aggr}}$?

  1. $f^+()$ and $f^-()$ are not expressible in $\mathcal{L}_{\mathrm{aggr}}$ if the attack graph of (2) is cyclic (because queries in $\mathcal{L}_{\mathrm{aggr}}$ are Hanf-local).
  2. Does the converse hold?

Formal Setting

- Numerical terms $f()$ expressible in the (safe) rule format

$$\texttt{AGG}(r) \leftarrow R_1(\underline{\vec{x}_1}, \vec{y}_1) \wedge R_2(\underline{\vec{x}_2}, \vec{y}_2) \wedge \cdots \wedge R_n(\underline{\vec{x}_n}, \vec{y}_n), \qquad (2)$$

  where r is either a numerical variable or a constant, and `AGG` is an aggregate operator (e.g., `MAX`, `MIN`, `SUM`, `COUNT`, `AVG`); assume $R_i \neq R_j$ if $i \neq j$.
- Given a database instance, let $f^{+}()$ and $f^{-}()$ be, respectively, the greatest and smallest values of $f()$ across all repairs.
- Aggregate logic $\mathcal{L}_{\mathrm{aggr}}$ [Hella et al., 2001]: FOL + aggregation.
- Question in [Fuxman, 2007] and [Dixit and Kolaitis, 2022]:

  When can $f^{+}()$ and $f^{-}()$ be expressed in $\mathcal{L}_{\mathrm{aggr}}$?

  1. $f^{+}()$ and $f^{-}()$ are not expressible in $\mathcal{L}_{\mathrm{aggr}}$ if the attack graph of (2) is cyclic (because queries in $\mathcal{L}_{\mathrm{aggr}}$ are Hanf-local).
  2. Does the converse hold?

Rewriting Example

$$\texttt{MAX}(z) \leftarrow \mathsf{ACTORS}(\underline{x}, \mathsf{F}, z).$$

- Upper bound rewriting:

$$\mathsf{UB}(\texttt{MAX}(z)) \leftarrow \mathsf{ACTORS}(\underline{x}, \mathsf{F}, z)$$

- Lower bound rewriting:

$$\begin{aligned}
\mathsf{POSSIBLE_M}(x) &\leftarrow \mathsf{ACTORS}(\underline{x}, \mathsf{M}, z)\\
\mathsf{CERTAIN_F}(x, z) &\leftarrow \mathsf{ACTORS}(\underline{x}, \mathsf{F}, z), \neg\mathsf{POSSIBLE_M}(x)\\
\mathsf{L}(x, \texttt{MIN}(z)) &\leftarrow \mathsf{CERTAIN_F}(\underline{x}, z)\\
\mathsf{LB}(\texttt{MAX}(z)) &\leftarrow \mathsf{L}(x, z)
\end{aligned}$$

Table of Contents

Counting

Given a Boolean query Q, define the following counting problem:

Problem $\sharp$CERTAINTY(Q)

Input: A database instance that may violate primary-key constraints.

Question: How many repairs of satisfy Q?

Complexity Classification Task

Input: A self-join-free Boolean conjunctive query Q.

Task: Determine lower and upper complexity bounds on the complexity of $\sharp$CERTAINTY(q), in terms of common complexity classes like FP and $\sharp$P.

- Solved in [Maslowski and W., 2013] and generalized to FDs in [Calautti et al., 2022].
- Same problem as query answering in block-independent disjoint (BID) probabilistic databases under the restriction that in every block $\mathbf{b}$, every tuple has probability $\frac{1}{|\mathbf{b}|}$.

Counting

Given a Boolean query Q, define the following counting problem:

Problem $\sharp$CERTAINTY(Q)

Input: A database instance that may violate primary-key constraints.

Question: How many repairs of satisfy Q?

Complexity Classification Task

Input: A self-join-free Boolean conjunctive query Q.

Task: Determine lower and upper complexity bounds on the complexity of $\sharp$CERTAINTY(q), in terms of common complexity classes like FP and $\sharp$P.

- Solved in [Maslowski and W., 2013] and generalized to FDs in [Calautti et al., 2022].
- Same problem as query answering in block-independent disjoint (BID) probabilistic databases under the restriction that in every block $\mathbf{b}$, every tuple has probability $\frac{1}{|\mathbf{b}|}$.

Counting

Given a Boolean query Q, define the following counting problem:

Problem $\sharp$CERTAINTY(Q)

Input: A database instance that may violate primary-key constraints.

Question: How many repairs of satisfy Q?

Complexity Classification Task

Input: A self-join-free Boolean conjunctive query Q.

Task: Determine lower and upper complexity bounds on the complexity of $\sharp$CERTAINTY(q), in terms of common complexity classes like FP and $\sharp$P.

- Solved in [Maslowski and W., 2013] and generalized to FDs in [Calautti et al., 2022].
- Same problem as query answering in block-independent disjoint (BID) probabilistic databases under the restriction that in every block $\mathbf{b}$, every tuple has probability $\frac{1}{|\mathbf{b}|}$.

BID Databases

Every input to CERTAINTY(Q) is a block-independent disjoint database without probabilities (or with uniform probabilities).

☞ Inconsistency is not only a burden, but also a chance. [1]

Researchers:

	Name	Affiliation	P
t_1^1	Fred	U. Washington	$p_1^1 = 0.3$
t_1^2		U. Wisconsin	$p_1^2 = 0.2$
t_1^3		Y! Research	$p_1^3 = 0.5$
t_2^1	Sue	U. Washington	$p_2^1 = 1.0$
t_3^1	John	U. Wisconsin	$p_3^1 = 0.7$
t_3^2		U. Washington	$p_3^2 = 0.3$
t_4^1	Frank	Y! Research	$p_4^1 = 0.9$
t_4^2		M. Research	$p_4^2 = 0.1$

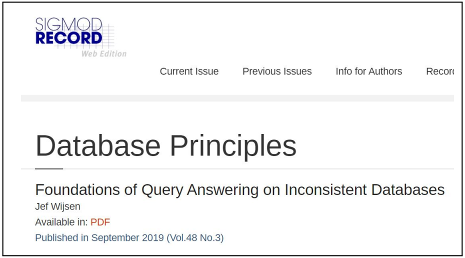
[1] Inspired by [Kern-Isberner and Lukasiewicz, 2017]. The image is from [Dalvi et al., 2009].

Table of Contents

Concluding Remarks

Consistent Query Answering is an active research area since [Arenas et al., 1999]:

- Database repairing w.r.t. different classes of constraints
- Database repairing and data exchange
- Database repairing and approximations
- Database repairing and preferences
- Database repairing and implementations
- Database repairing and database management systems
- Consistent query answering for queries with negation
- Consistent query answering in description logics
- Consistent query answering over graph databases
- ...


SIGMOD RECORD
Web Edition

Current Issue Previous Issues Info for Authors Recor

Database Principles

Foundations of Query Answering on Inconsistent Databases
Jef Wijsen
Available in: PDF
Published in September 2019 (Vol.48 No.3)

Communications of the ACM, March 2024

research

Check for updates

DOI:10.1145/3624717

Deploying possible world semantics and the challenge of computing the certain answers to queries.

BY BENNY KIMELFELD AND PHOKION G. KOLAITIS

A Unifying Framework for Incompleteness, Inconsistency, and Uncertainty in Databases

DATABASES ARE OFTEN assumed to have definite content. The reality, though, is the database at hand may be deficient due to missing, invalid, or uncertain information. As a simple illustration, the

may invol
computat
rors in sta
models, c
that even
in full cc
to queries
may depe
rectify it;
may vary
even tho
equally leg
In the
tions, the
research c
tal appro
deficiency
trariness.
article is t
in which
has been
proach to
ciency in
approach,
need to r
missing
records (h
rect erron
since ma
and since
correct on
cific one. I
database

» key

- Possible
queries c
uncertai
viewed a
their pos
answers
true in ev
- Queries t
tractable
semantic

Thanks!

FYI, Brad Pitt celebrated his 60th birthday on December 18, 2023.

References I

Arenas, M., Bertossi, L. E., and Chomicki, J. (1999).
Consistent query answers in inconsistent databases.
In *PODS*, pages 68–79. ACM Press.

Arenas, M., Bertossi, L. E., and Chomicki, J. (2001).
Scalar aggregation in fd-inconsistent databases.
In *ICDT*, volume 1973 of *Lecture Notes in Computer Science*, pages 39–53. Springer.

Calautti, M., Livshits, E., Pieris, A., and Schneider, M. (2022).
Counting database repairs entailing a query: The case of functional dependencies.
In *PODS*, pages 403–412. ACM.

Chomicki, J. and Marcinkowski, J. (2005).
Minimal-change integrity maintenance using tuple deletions.
Inf. Comput., 197(1-2):90–121.

Dalvi, N. N., Ré, C., and Suciu, D. (2009).
Probabilistic databases: diamonds in the dirt.
Commun. ACM, 52(7):86–94.

Dixit, A. A. and Kolaitis, P. G. (2022).
Consistent answers of aggregation queries via SAT.
In *ICDE*, pages 924–937. IEEE.

Fan, Z., Koutris, P., Ouyang, X., and Wijsen, J. (2023).
LinCQA: Faster consistent query answering with linear time guarantees.
Proc. ACM Manag. Data, 1(1):38:1–38:25.

Fontaine, G. (2015).
Why is it hard to obtain a dichotomy for consistent query answering?
ACM Trans. Comput. Log., 16(1):7:1–7:24.

References II

Fuxman, A. (2007).
Efficient query processing over inconsistent databases.
PhD thesis, University of Toronto.

Hella, L., Libkin, L., Nurmonen, J., and Wong, L. (2001).
Logics with aggregate operators.
J. ACM, 48(4):880–907.

Kern-Isberner, G. and Lukasiewicz, T. (2017).
Many facets of reasoning under uncertainty, inconsistency, vagueness, and preferences: A brief survey.
KI, 31(1):9–13.

Koutris, P., Ouyang, X., and Wijsen, J. (2021).
Consistent query answering for primary keys on path queries.
In *PODS*, pages 215–232. ACM.

Koutris, P. and W., J. (2017).
Consistent query answering for self-join-free conjunctive queries under primary key constraints.
ACM Trans. Database Syst., 42(2):9:1–9:45.

Maslowski, D. and W., J. (2013).
A dichotomy in the complexity of counting database repairs.
J. Comput. Syst. Sci., 79(6):958–983.

Padmanabha, A., Segoufin, L., and Sirangelo, C. (2023).
A dichotomy in the complexity of consistent query answering for two atom queries with self-join.
CoRR, abs/2309.12059.

W., J. (2010).
A remark on the complexity of consistent conjunctive query answering under primary key violations.
Inf. Process. Lett., 110(21):950–955.