Applications of knowledge compilation to databases counting/probabilistic problems

Mikaël Monet

May 26th, 2026

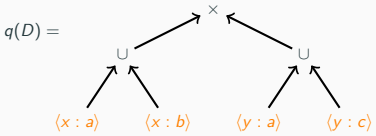
Dagstuhl seminar #26221: *Knowledge Compilation in Artificial Intelligence, Databases, and Formal Methods*

Introduction

Dan Olteanu's talk: using **multi-valued** circuits (a.k.a. *factorized databases*) to represent query results

$q(x, y) =$ a query with **free vars**

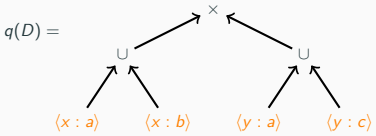
$D =$ a database



Introduction

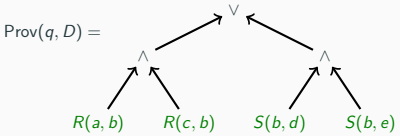
Dan Olteanu's talk: using **multi-valued** circuits (a.k.a. *factorized databases*) to represent query results

$q(x, y)$ = a query with **free vars**
D = a database



This talk: using Boolean circuits for problems that can be solved via *Boolean provenance*

q: a Boolean query
D = a database



Outline

1. Boolean provenance
2. Probabilistic databases
   - Compiling hierarchical conjunctive queries
   - Compiling MSO queries on tree-like data
3. Other problems: Shapley values, resilience
4. ProvSQL teaser

Boolean provenance

Provenance

- A (relational) database D: a finite set of *facts/tuples*
  - → **Example:** $D = \{R(a, b), R(b, c), S(b, b, a)\}$
- A Boolean query q: a function that maps databases to **true** or **false**
  - → **Example:** $q =$ "the database contains at least 10 facts"

Provenance

- A (relational) database D: a finite set of *facts/tuples*
  - → **Example:** $D = \{R(a, b), R(b, c), S(b, b, a)\}$
- A Boolean query q: a function that maps databases to **true** or **false**
  - → **Example:** $q =$ "the database contains at least 10 facts"

Definition [Imielinski & Lipski 1984]

The Boolean provenance $\mathrm{Prov}(q, D)$ is the Boolean function with facts of D as variables and such that, for any $D' \subseteq D$, $\mathrm{Prov}(q, D)$ evaluates to true on D' if and only if $D' \models q$

Boolean provenance: example

$D =$

Likes	
Alice	Bob
Alice	John
Bob	Bob
John	Bob

$q = \exists x, y, z : L(x, z) \wedge L(y, z) \wedge x \neq y$
("there are two people who like the same person")

Boolean provenance: example

$D =$

Likes	
Alice	Bob
Alice	John
Bob	Bob
John	Bob

$q = \exists x, y, z : L(x, z) \wedge L(y, z) \wedge x \neq y$
("there are two people who like the same person")

$$\begin{aligned}\mathrm{Prov}(q, D) = &[L(A, B) \wedge L(B, B)] \\ &\vee [L(A, B) \wedge L(J, B)] \\ &\vee [L(B, B) \wedge L(J, B)]\end{aligned}$$

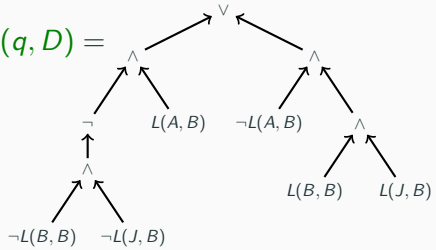
Boolean provenance: example

$D =$

Likes	
Alice	Bob
Alice	John
Bob	Bob
John	Bob

$q = \exists x, y, z : L(x, z) \wedge L(y, z) \wedge x \neq y$
("there are two people who like the same person")

$\mathrm{Prov}(q, D) =$

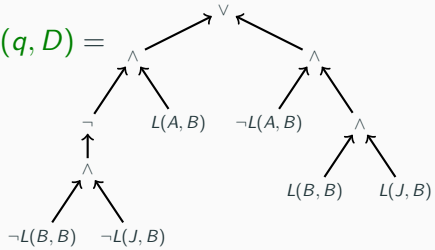


Boolean provenance: example

$D =$

Likes	
Alice	Bob
Alice	John
Bob	Bob
John	Bob

$\mathrm{Prov}(q, D) =$



$q = \exists x, y, z : L(x, z) \wedge L(y, z) \wedge x \neq y$
("there are two people who like the same person")

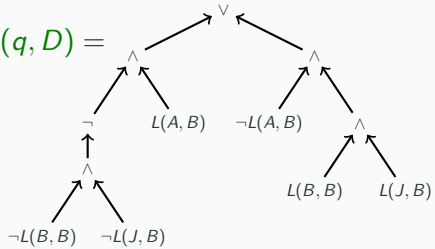
If q is an FO sentence ($\simeq$ an SQL Boolean query), what is the data complexity of computing $\mathrm{Prov}(q, D)$?

Boolean provenance: example

$D =$

Likes	
Alice	Bob
Alice	John
Bob	Bob
John	Bob

$\mathrm{Prov}(q, D) =$



$q = \exists x, y, z : L(x, z) \wedge L(y, z) \wedge x \neq y$
("there are two people who like the same person")

If q is an FO sentence ($\simeq$ an SQL Boolean query), what is the data complexity of computing $\mathrm{Prov}(q, D)$? PTIME

Probabilistic databases

Tuple-independent probabilistic databases

- **Probabilistic databases**: to represent data uncertainty
  → simplest formalism: tuple-independent database

$D =$

Likes		p
Alice	Bob	0.5
Alice	John	1
Bob	Bob	0.2
John	Bob	0.7

Tuple-independent probabilistic databases

- **Probabilistic databases**: to represent data uncertainty
  → simplest formalism: tuple-independent database

$D' =$

Likes		p
		0.5
Alice	John	1
		0.2
John	Bob	0.7

Tuple-independent probabilistic databases

- **Probabilistic databases**: to represent data uncertainty
  → simplest formalism: tuple-independent database

$D' =$

Likes		p
Alice	John	0.5
John	Bob	1
		0.2
		0.7

$\Pr(D') = (1 - 0.5) \times 1 \times (1 - 0.2) \times 0.7$

Tuple-independent probabilistic databases

- **Probabilistic databases**: to represent data uncertainty
  → simplest formalism: tuple-independent database

$D =$

Likes		p
Alice	Bob	0.5
Alice	John	1
Bob	Bob	0.2
John	Bob	0.7

$q =$ "there are two people who like the same person"
$\exists x, y, z : L(x, z) \wedge L(y, z) \wedge x \neq y$

Tuple-independent probabilistic databases

- **Probabilistic databases**: to represent data uncertainty
  → simplest formalism: tuple-independent database

$D =$

Likes		p
Alice	Bob	0.5
Alice	John	1
Bob	Bob	0.2
John	Bob	0.7

$q =$ "there are two people who like the same person"
$\exists x, y, z : L(x, z) \wedge L(y, z) \wedge x \neq y$

$$\Pr(D \models q) = \sum_{\substack{D' \subseteq D \\ D' \models q}} \Pr(D')$$

Tuple-independent probabilistic databases

- **Probabilistic databases**: to represent data uncertainty
  → simplest formalism: tuple-independent database

$D =$

Likes		p
Alice	Bob	0.5
Alice	John	1
Bob	Bob	0.2
John	Bob	0.7

$q =$ "there are two people who like the same person"
$\exists x, y, z : L(x, z) \wedge L(y, z) \wedge x \neq y$

$\Pr(D \models q) = \sum_{\substack{D' \subseteq D \\ D' \models q}} \Pr(D')$ (**not efficient**)

Tuple-independent probabilistic databases

- **Probabilistic databases**: to represent data uncertainty
  → simplest formalism: tuple-independent database

$D =$

Likes		p
Alice	Bob	0.5
Alice	John	1
Bob	Bob	0.2
John	Bob	0.7

$q =$ "there are two people who like the same person"
$\exists x, y, z : L(x, z) \wedge L(y, z) \wedge x \neq y$

Queries with free variables? $q(s, t) = \ldots$

$\Pr(D \models q) = \sum_{\substack{D' \subseteq D \\ D' \models q}} \Pr(D')$ (**not efficient**)

Boolean provenance to the rescue!

$D =$

Likes		p
Alice	Bob	0.5
Alice	John	1
Bob	Bob	0.2
John	Bob	0.7

$q = \exists x, y, z : L(x, z) \wedge L(y, z) \wedge x \neq y$

Boolean provenance to the rescue!

$D =$

Likes		p
Alice	Bob	0.5
Alice	John	1
Bob	Bob	0.2
John	Bob	0.7

$$\mathrm{Prov}(q, D) = [L(A,B) \wedge L(B,B)] \vee [L(A,B) \wedge L(J,B)] \vee [L(B,B) \wedge L(J,B)]$$

$$q = \exists x, y, z : L(x,z) \wedge L(y,z) \wedge x \neq y$$

Boolean provenance to the rescue!

$D =$

Likes		p
Alice	Bob	0.5
Alice	John	1
Bob	Bob	0.2
John	Bob	0.7

$$\begin{aligned}\mathrm{Prov}(q, D) = &[L(A, B) \wedge L(B, B)] \\ \vee &[L(A, B) \wedge L(J, B)] \\ \vee &[L(B, B) \wedge L(J, B)]\end{aligned}$$

$q = \exists x, y, z : L(x, z) \wedge L(y, z) \wedge x \neq y$

We have $\Pr(D \models q) = \Pr(\mathrm{Prov}(q, D) = \mathrm{true})$

Boolean provenance to the rescue!

$D =$

Likes		p
Alice	Bob	0.5
Alice	John	1
Bob	Bob	0.2
John	Bob	0.7

$$\begin{aligned}\mathrm{Prov}(q, D) = &[L(A,B) \wedge L(B,B)] \\ \vee &[L(A,B) \wedge L(J,B)] \\ \vee &[L(B,B) \wedge L(J,B)]\end{aligned}$$

$q = \exists x, y, z : L(x, z) \wedge L(y, z) \wedge x \neq y$

We have $\Pr(D \models q) = \Pr(\mathrm{Prov}(q, D) = \mathrm{true})$

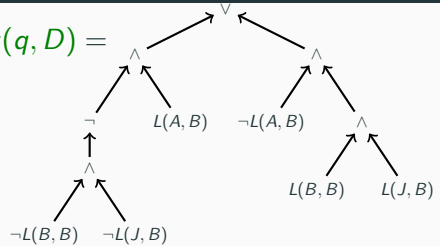
Computing the probability of an arbitrary circuit is #P-hard!

Boolean provenance to the rescue!

$D =$

Likes		p
Alice	Bob	0.5
Alice	John	1
Bob	Bob	0.2
John	Bob	0.7

$\text{Prov}(q, D) =$



$q = \exists x, y, z : L(x, z) \wedge L(y, z) \wedge x \neq y$

We have $\Pr(D \models q) = \Pr(\text{Prov}(q, D) = \text{true})$

Computing the probability of an arbitrary circuit is #P-hard!

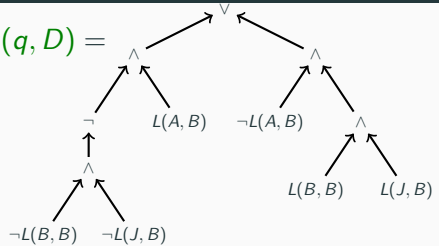
What if we can compute a **decomposable deterministic circuit (d-D)** efficiently?

Boolean provenance to the rescue!

$D =$

Likes		p
Alice	Bob	0.5
Alice	John	1
Bob	Bob	0.2
John	Bob	0.7

$\text{Prov}(q, D) =$



$q = \exists x, y, z : L(x, z) \wedge L(y, z) \wedge x \neq y$

We have $\Pr(D \models q) = \Pr(\text{Prov}(q, D) = \text{true})$

Computing the probability of an arbitrary circuit is #P-hard!

What if we can compute a **decomposable deterministic circuit (d-D)** efficiently?

$\Longrightarrow$ PTIME

The probabilistic query evaluation problem ($\mathrm{PQE}(q)$)

Definition: problem $\mathrm{PQE}(q)$, for q a Boolean query

Input: a tuple-independent probabilistic database D

Output: $\mathrm{Pr}(D \models q)$

The probabilistic query evaluation problem ($\mathrm{PQE}(q)$)

Definition: problem $\mathrm{PQE}(q)$, for q a Boolean query

Input: a tuple-independent probabilistic database D

Output: $\Pr(D \models q)$

- Dalvi and Suciu [JACM'12] have shown a **dichotomy** on the (data) complexity of $\mathrm{PQE}(q)$ for unions of conjunctive queries:
  - either $\mathrm{PQE}(q) \in \mathrm{PTIME}$, and q is called "safe"
  - or $\mathrm{PQE}(q)$ is $\mathrm{FP}^{\#\mathrm{P}}$**-hard**, and q is called "unsafe"

The probabilistic query evaluation problem ($\mathrm{PQE}(q)$)

Definition: problem $\mathrm{PQE}(q)$, for q a Boolean query

Input: a tuple-independent probabilistic database D

Output: $\Pr(D \models q)$

- Dalvi and Suciu [JACM'12] have shown a **dichotomy** on the (data) complexity of $\mathrm{PQE}(q)$ for unions of conjunctive queries:
  - either $\mathrm{PQE}(q) \in \mathrm{PTIME}$, and q is called "safe"
  - or $\mathrm{PQE}(q)$ is $\mathrm{FP}^{\#\mathrm{P}}$**-hard**, and q is called "unsafe"
- Their algorithm for a safe query q essentially uses three rules:

  $\rightarrow$ Independence: $\Pr(A \wedge B) = \Pr(A) \times \Pr(B)$ when A, B are independent events

The probabilistic query evaluation problem ($\mathrm{PQE}(q)$)

Definition: problem $\mathrm{PQE}(q)$, for q a Boolean query

Input: a tuple-independent probabilistic database D

Output: $\Pr(D \models q)$

- Dalvi and Suciu [JACM'12] have shown a **dichotomy** on the (data) complexity of $\mathrm{PQE}(q)$ for unions of conjunctive queries:
  - either $\mathrm{PQE}(q) \in \mathrm{PTIME}$, and q is called "safe"
  - or $\mathrm{PQE}(q)$ is $\mathrm{FP}^{\#\mathrm{P}}$**-hard**, and q is called "unsafe"

- Their algorithm for a safe query q essentially uses three rules:
  - $\rightarrow$ Independence: $\Pr(A \wedge B) = \Pr(A) \times \Pr(B)$ when A, B are independent events
  - $\rightarrow$ Negation: $\Pr(\neg A) = 1 - \Pr(A)$

The probabilistic query evaluation problem ($\mathrm{PQE}(q)$)

Definition: problem $\mathrm{PQE}(q)$, for q a Boolean query

Input: a tuple-independent probabilistic database D

Output: $\Pr(D \models q)$

- Dalvi and Suciu [JACM'12] have shown a **dichotomy** on the (data) complexity of $\mathrm{PQE}(q)$ for unions of conjunctive queries:
  - either $\mathrm{PQE}(q) \in \mathrm{PTIME}$, and q is called "safe"
  - or $\mathrm{PQE}(q)$ is $\mathrm{FP}^{\#\mathrm{P}}$**-hard**, and q is called "unsafe"

- Their algorithm for a safe query q essentially uses three rules:
  - → Independence: $\Pr(A \wedge B) = \Pr(A) \times \Pr(B)$ when A, B are independent events
  - → Negation: $\Pr(\neg A) = 1 - \Pr(A)$
  - → Inclusion–exclusion:
    $\Pr(A \vee B \vee C \vee \ldots) = \Pr(A) + \Pr(B) + \ldots - \Pr(A \wedge B) - \Pr(A \wedge C) - \ldots + \Pr(A \wedge B \wedge C) + \ldots$

An open problem

- If q is a safe UCQ (i.e., $\mathrm{PQE}(q)$ is PTIME), can we compute $\mathrm{Prov}(q, D)$ as a d-D in PTIME?
  - $\rightarrow$ This is not known! (See [this arXiv note] for a reformulation as a conjecture about inclusion-exclusion)

An open problem

- If q is a safe UCQ (i.e., $\mathrm{PQE}(q)$ is PTIME), can we compute $\mathrm{Prov}(q, D)$ as a d-D in PTIME?
  - $\rightarrow$ This is not known! (See [this arXiv note] for a reformulation as a conjecture about inclusion-exclusion)

- But we know this can be done, for instance, when q is a hierarchical self-join free conjunctive query
  - $\rightarrow$ We prove this next

Hierarchical self-join free conjunctive query

- A conjunctive query (CQ): an existentially quantified conjunction of atoms
  $\rightarrow$ **Example:** $q = \exists x\, y : R(x, y) \wedge R(y, x) \wedge S(x, x)$

Hierarchical self-join free conjunctive query

- A conjunctive query (CQ): an existentially quantified conjunction of atoms
  → **Example:** $q = \exists x\, y : R(x, y) \wedge R(y, x) \wedge S(x, x)$
- self-join free CQ: no repetition of relational symbols
  → **Example:** $q = \exists x\, y : R(x, y) \wedge S(y, x) \wedge T(x, x)$

Hierarchical self-join free conjunctive query

- A conjunctive query (CQ): an existentially quantified conjunction of atoms
  → **Example:** $q = \exists x\, y : R(x, y) \wedge R(y, x) \wedge S(x, x)$
- self-join free CQ: no repetition of relational symbols
  → **Example:** $q = \exists x\, y : R(x, y) \wedge S(y, x) \wedge T(x, x)$

Theorem [Dalvi & Suciu 2004]

Let q be a Boolean self-join free CQ. If q is hierarchical then $\mathrm{PQE}(q)$ is PTIME, otherwise it is #P-hard

Hierarchical self-join free conjunctive query

- A conjunctive query (CQ): an existentially quantified conjunction of atoms
  → **Example:** $q = \exists x\, y : R(x, y) \wedge R(y, x) \wedge S(x, x)$
- self-join free CQ: no repetition of relational symbols
  → **Example:** $q = \exists x\, y : R(x, y) \wedge S(y, x) \wedge T(x, x)$

Theorem [Dalvi & Suciu 2004]

Let q be a Boolean self-join free CQ. If q is hierarchical then $\mathrm{PQE}(q)$ is PTIME, otherwise it is #P-hard

- **Definition:** q is hierarchical if, for every $x, y \in \mathrm{vars}(q)$, we have either $\mathrm{atoms}_q(x) \subseteq \mathrm{atoms}_q(y)$, or $\mathrm{atoms}_q(x) \subseteq \mathrm{atoms}_q(y)$, or $\mathrm{atoms}_q(x) \cap \mathrm{atoms}_q(y) = \emptyset$

Hierarchical self-join free conjunctive query

- A conjunctive query (CQ): an existentially quantified conjunction of atoms
  → **Example:** $q = \exists x\, y : R(x, y) \wedge R(y, x) \wedge S(x, x)$
- self-join free CQ: no repetition of relational symbols
  → **Example:** $q = \exists x\, y : R(x, y) \wedge S(y, x) \wedge T(x, x)$

Theorem [Dalvi & Suciu 2004]

Let q be a Boolean self-join free CQ. If q is hierarchical then $\mathrm{PQE}(q)$ is PTIME, otherwise it is #P-hard

- **Definition:** q is hierarchical if, for every $x, y \in \mathrm{vars}(q)$, we have either $\mathrm{atoms}_q(x) \subseteq \mathrm{atoms}_q(y)$, or $\mathrm{atoms}_q(x) \subseteq \mathrm{atoms}_q(y)$, or $\mathrm{atoms}_q(x) \cap \mathrm{atoms}_q(y) = \emptyset$

→ Proof of this on board, using d-Ds for the tractability side ←

Monadic second-order logic on treelike data (1/2)

What if q is not safe? Can we recover tractability by restricting the set of databases?

Monadic second-order logic on treelike data (1/2)

What if q is not safe? Can we recover tractability by restricting the set of databases?

- Monadic second-order logic (MSO): like FO but we have monadic (representing sets of elements) variables
  → **Example:**

$$
\begin{aligned}
q = \exists X : (&\text{Bob} \in X) \\
&\wedge (\forall y, z : y \in X \wedge \text{Likes}(y, z) \implies z \in X) \\
&\wedge \forall X' : [(\text{Bob} \in X') \\
&\qquad \wedge (\forall y, z : y \in X' \wedge \text{Likes}(y, z) \implies z \in X' \\
&\qquad \implies X \subseteq X') \\
&\wedge \text{Alice} \in X
\end{aligned}
$$

Monadic second-order logic on treelike data (1/2)

What if q is not safe? Can we recover tractability by restricting the set of databases?

- Monadic second-order logic (MSO): like FO but we have monadic (representing sets of elements) variables
  → **Example:**

$$
\begin{aligned}
q = \exists X : (&\text{Bob} \in X) \\
&\wedge (\forall y, z : y \in X \wedge \text{Likes}(y, z) \implies z \in X) \\
&\wedge \forall X' : [(\text{Bob} \in X') \\
&\qquad \wedge (\forall y, z : y \in X' \wedge \text{Likes}(y, z) \implies z \in X' \\
&\qquad \implies X \subseteq X') \\
&\wedge \text{Alice} \in X
\end{aligned}
$$

- Treewidth of a database: a measure indicating how close the database is to a tree

Monadic second-order logic on treelike data (2/2)

Theorem [Amarilli, Bourhis & Senellart 2016]

Let q be a Boolean MSO query, and $k \in \mathbb{N}$. Then $\mathrm{PQE}(q)$ is PTIME when restricted to databases of treewidth $\leq k$. Moreover we can build d-Ds in PTIME representing the provenance.

Monadic second-order logic on treelike data (2/2)

Theorem [Amarilli, Bourhis & Senellart 2016]

Let q be a Boolean MSO query, and $k \in \mathbb{N}$. Then $\mathrm{PQE}(q)$ is PTIME when restricted to databases of treewidth $\leq k$. Moreover we can build d-Ds in PTIME representing the provenance.

$\rightarrow$ Proof (sketch) of this on board, using tree automata to build d-Ds $\leftarrow$

Other problems: Shapley values, resilience

Shapley values (1/2)

- q a Boolean query, D a database, $f \in D$ a fact of D
- We want to measure the contribution of f to $q(D)$

Shapley values (1/2)

- q a Boolean query, D a database, $f \in D$ a fact of D
- We want to measure the contribution of f to $q(D)$

  $\rightarrow$ We can use the established notion of Shapley value from cooperative game theory

Definition [Livshits, Bertossi, Kimelfeld, & Sebag 2020]

$$\text{Shapley}(q, D, f) = \sum_{S \subseteq D \setminus \{f\}} \frac{|S|!(|D| - |S| - 1)!}{|D|!} \big(q(S \cup \{f\}) - q(S)\big)$$

Shapley values (2/2)

Theorem [Deutch, Frost, Kimelfeld & Monet 2022]

- for any Boolean query q, computing Shapley values for q reduces in PTIME to $\mathrm{PQE}(q)$
- we can compute in PTIME the Shapley values of d-D circuits (bottom-up dynamic programming)

Shapley values (2/2)

Theorem [Deutch, Frost, Kimelfeld & Monet 2022]

- for any Boolean query q, computing Shapley values for q reduces in PTIME to $\mathrm{PQE}(q)$
- we can compute in PTIME the Shapley values of d-D circuits (bottom-up dynamic programming)

- There is also a notion of Shapley value for probabilistic databases!
  - → See Karmakar, Monet, Senellart & Bressan 2024. In this case the problem is PTIME equivalent to PQE.

→ We can compute in PTIME (probabilistic or deterministic) Shapley values for safe queries on all database, or for MSO queries on tree-like data

Resilience

- q a Boolean query, D a database
- $\mathrm{Res}(q, D) = \min_{\substack{D' \subseteq D \\ s.t. D \setminus D' \not\models q}} |D'|$. In English: the minimum number of facts to remove from D to make the query false.

Resilience

- q a Boolean query, D a database
- $\text{Res}(q, D) = \min_{\substack{D' \subseteq D \\ s.t. D \setminus D' \not\models q}} |D'|$. In English: the minimum number of facts to remove from D to make the query false.
- Equivalently: the size of the **biggest** subdatabase of D that make **the negation of the query** true

Resilience

- q a Boolean query, D a database
- $\mathrm{Res}(q, D) = \min_{\substack{D' \subseteq D \\ s.t. D \setminus D' \not\models q}} |D'|$. In English: the minimum number of facts to remove from D to make the query false.
- Equivalently: the size of the **biggest** subdatabase of D that make **the negation of the query** true
- on board: computing the maximum Hamming weight of a satisfying assignement of a d-DNNF

Resilience

- q a Boolean query, D a database
- $\mathrm{Res}(q, D) = \min_{\substack{D' \subseteq D \\ s.t. D \setminus D' \not\models q}} |D'|$. In English: the minimum number of facts to remove from D to make the query false.
- Equivalently: the size of the **biggest** subdatabase of D that make **the negation of the query** true
- on board: computing the maximum Hamming weight of a satisfying assignement of a d-DNNF

$\rightarrow$ We can compute resilience in PTIME if we can build d-DNNFs for $\neg q$

ProvSQL teaser

ProvSQL

- ProvSQL: a PostgreSQL module to compute provenance, lead by Pierre Senellart
- Can handle other semirings than the Boolean semiring
- To solve PQE/Shapley: first compute $\mathrm{Prov}(q, D)$ as a Boolean circuit, then either
  - call a **knowledge compiler** to compile to d-DNNF
  - if the circuit's **treewidth** is small, convert it to d-DNNF using [Amarilli, Capelli, Monet & Senellart 2020]

ProvSQL

- ProvSQL: a PostgreSQL module to compute provenance, lead by Pierre Senellart
- Can handle other semirings than the Boolean semiring
- To solve PQE/Shapley: first compute Prov(q, D) as a Boolean circuit, then either
  - call a **knowledge compiler** to compile to d-DNNF
  - if the circuit's **treewidth** is small, convert it to d-DNNF using [Amarilli, Capelli, Monet & Senellart 2020]

→ See **Pierre's demo on Friday**!

Thanks for your attention!

Bibliography i


Antoine Amarilli, Pierre Bourhis, and Pierre Senellart.
Tractable lineages on treelike instances: Limits and extensions.
In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 355–370, 2016.


Antoine Amarilli, Florent Capelli, Mikaël Monet, and Pierre Senellart.
Connecting knowledge compilation classes width parameters.
Theory of Computing Systems, 64(5):861–914, 2020.


Antoine Amarilli, Mikaël Monet, and Dan Suciu.
The non-cancelling intersections conjecture.
CoRR, abs/2401.16210, 2024.

Bibliography ii


Nilesh N. Dalvi and Dan Suciu.
Efficient query evaluation on probabilistic databases.
In *VLDB*, pages 864–875. Morgan Kaufmann, 2004.


Nilesh N. Dalvi and Dan Suciu.
The dichotomy of probabilistic inference for unions of conjunctive queries.
Journal of the ACM, 59(6):30, 2012.


Daniel Deutch, Nave Frost, Benny Kimelfeld, and Mikaël Monet.
Computing the shapley value of facts in query answering.
In *Proceedings of the 2022 International Conference on Management of Data*, pages 1570–1583, 2022.

Tomasz Imieliński and Witold Lipski Jr.
Incomplete information in relational databases.
Journal of the ACM (JACM), 31(4):761–791, 1984.

Pratik Karmakar, Mikaël Monet, Pierre Senellart, and Stéphane Bressan.
Expected shapley-like scores of boolean functions: Complexity and applications to probabilistic databases.
Proceedings of the ACM on Management of Data, 2(2):1–26, 2024.