A Tutorial on Query Answering and Reasoning over Probabilistic Knowledge Bases*

İsmail İlkan Ceylan and Thomas Lukasiewicz

University of Oxford, Oxford, UK
firstname.lastname@cs.ox.ac.uk

Abstract. Large-scale probabilistic knowledge bases are becoming increasingly important in academia and industry alike. They are constantly extended with new data, powered by modern information extraction tools that associate probabilities with knowledge base facts. This tutorial is dedicated to give an understanding of various query answering and reasoning tasks that can be used to exploit the full potential of probabilistic knowledge bases. In the first part of the tutorial, we focus on (tuple-independent) probabilistic databases as the simplest probabilistic data model. In the second part of the tutorial, we move on to richer representations where the probabilistic database is extended with ontological knowledge. For each part, we review some known data complexity results as well as discuss some recent results.

Keywords: Probabilistic Reasoning · Probabilistic Databases · Probabilistic Knowledge Bases · Query Answering · Data Complexity.

1 Introduction

In recent years, there has been a strong interest in building *large-scale probabilistic knowledge bases* from data in an automated way, which has resulted in a number of systems, such as DeepDive [65], NELL [50], Reverb [27], Yago [40], Microsoft's Probase [75], IBM's Watson [29], and Google's Knowledge Vault [25]. These systems continuously crawl the Web and extract *structured information*, and thus populate their databases with millions of entities and billions of tuples. To what extent can these search and extraction systems help with real-world use cases? This turns out to be an open-ended question. For example, DeepDive is used to build knowledge bases for domains such as paleontology, geology, medical genetics, and human movement [46, 55]; it also serves as an important tool for the fight against human trafficking [35]. IBM's Watson makes impact on health-care systems [30] and many other application domains of life sciences. Google's Knowledge Vault has compiled more than a billion facts from the Web and is primarily used to improve the quality of search results on the Web [26],

* This tutorial is mostly based on the dissertation work [13], and previously published material [8, 14], and also makes use of some material from the classical literature on probabilistic databases [68, 21].

From a broader perspective, the quest for building large knowledge bases serves as a new dawn for artificial intelligence (AI) research in general and for unifying logic and probability in particular. Fields such as information extraction, natural language processing (e.g., question answering), relational and deep learning, knowledge representation and reasoning, and databases are taking initiative towards a common goal.

The most basic model to manage, store, and process large-scale probabilistic data is that of *tuple-independent probabilistic databases* (PDBs) [68], which indeed underlies many of these systems [25, 65]. The first part of this tutorial is thus dedicated to give an overview on PDBs. We first recall the basics of query answering over PDBs, and then give an overview of the data complexity dichotomy between polynomial time and #P for evaluating unions of conjunctive queries over PDBs [21]. Then, we focus on two additional inference tasks that are inspired by *maximal posterior probability* computations in probabilistic graphical models (PGMs) [44]. That is, we discuss the problems of finding the *most probable database* and the *most probable hypothesis* for a given query, which intuitively correspond to finding explanations for PDB queries [37, 14].

Probabilistic databases typically lack a suitable handling of incompleteness, in practice. In particular, each of the above systems encodes only a portion of the real-world, and this description is necessarily *incomplete*. However, when it comes to querying, most of these systems employ the *closed-world assumption* (CWA) [60], i.e., any fact that is not present in the knowledge base is assigned the probability 0, and thus assumed to be impossible. It is also common practice to view every extracted fact as an *independent* Bernoulli variable, i.e., any two facts are probabilistically independent. Similarly, by the *closed-domain assumption (CDA)* of PDBs, the domain of discourse is fixed to a *finite* set of *known* constants, i.e., it is assumed that all individuals are known a priori. These assumptions are very strong, and lead to more problematic semantic consequences, once combined with another limitation of these systems, namely, the lack of *commonsense knowledge*, which brings us to probabilistic knowledge bases.

In the second part of this tutorial, we focus on *probabilistic knowledge bases*, which are probabilistic databases that additionally allow to encode commonsense knowledge. Note that incorporating commonsense knowledge is inherently connected to giving up the above completeness assumptions of standard PDBs. In the scope of this tutorial, we assume that commonsense knowledge is encoded in the form of *ontologies*. There are many different models that allow for encoding commonsense knowledge; see e.g. [36], but most of these models, such as MLNs [61], relational Bayesian networks [42], and function-free variants of probabilistic logic programs employ the closed-domain assumption, and therefore, do not allow fully fledged first-order knowledge, as it already occurs in (rather restricted) ontology languages. These semantic differences have been recently highlighted in a survey [9].

Ontologies are first-order theories that formalize domain-specific knowledge, thereby allowing for *open-world*, *open-domain* reasoning. The most prominent ontology languages in the literature are based on description logics (DLs) [3], and

lately also on Datalog$^{\pm}$ [12, 11, 10] (also studied as *existential rules*). For a uniform syntactic presentation, we focus on Datalog$^{\pm}$ ontologies, but we note that, in almost all cases, it is straight-forward to extend all the presented techniques and results. Interpreting databases under commonsense knowledge in the form of ontologies is closely related to ontology-based data access [56], also known as ontology-mediated query answering (OMQA) [7]. In the second part of this tutorial, we lift the reasoning problems introduced for probabilistic databases to ontology-mediated queries, and give an overview of various data complexity results.

2 Preliminaries: Logic, Databases, and Complexity

Intellectual roots of databases are in first-order logic [1]; in particular, in finite model theory [47]. Thus, we adopt the model-theoretic perspective and view databases as first-order structures over some fixed domain.

2.1 Logic and Notation

A *relational vocabulary* σ consists of sets $\mathbf{R}$ of *predicates*, $\mathbf{C}$ of *constants*, and $\mathbf{V}$ of *variable names* (or simply *variables*). The function $\mathsf{ar} : \mathbf{R} \mapsto \mathbb{N}$ associates with each predicate $\mathsf{P} \in \mathbf{R}$ a natural number, which defines the (unique) arity of P. A *term* is either a constant or a variable. An *atom* is of the form $\mathsf{P}(s_1, \ldots, s_n)$, where P is an n-ary predicate, and $s_1, \ldots, s_n$ are terms. A *ground atom* is an atom without variables.

A *first-order formula* is defined inductively by combining the logical atoms with logical connectives $\neg$, $\wedge$, $\vee$, and quantifiers $\exists$, $\forall$, as usual. A *literal* is either an atom or its negation. A *quantifier-free* formula is a formula that does not use a quantifier. A variable x in a formula Φ is *quantified*, or *bound*, if it is in the scope of a quantifier; otherwise, it is *free*. A *(first-order) sentence* is a first-order formula without any free variables, also called a *closed formula*, or *Boolean formula*. A *(first-order) theory* is a set of first-order sentences.

Let FO be the class of first-order formulas. The class of *existential first-order formulas* ($\exists\mathsf{FO}$) consists of first-order formulas of the form $\exists \boldsymbol{x}.\Phi(\boldsymbol{x})$, where Φ is any Boolean combination of atoms. The class of *universal first-order formulas* ($\forall\mathsf{FO}$) consists of first-order formulas of the form $\forall \boldsymbol{x}.\Phi(\boldsymbol{x})$, where Φ is any Boolean combination of atoms. A *disjunctive clause* is a finite disjunction of literals. A *conjunctive clause* is a finite conjunction of literals. The class of formulas in *existential conjunctive normal form* ($\exists\mathsf{CNF}$) consists of first-order formulas of the form $\exists \boldsymbol{x}.\Phi(\boldsymbol{x})$; the class of formulas in *universal conjunctive normal form* ($\forall\mathsf{CNF}$) consists of first-order formulas of the form $\forall \boldsymbol{x}.\Phi(\boldsymbol{x})$, where Φ is a conjunction of disjunctive clauses. The class of formulas in *existential disjunctive normal form* ($\exists\mathsf{DNF}$) consists of formulas of the form $\exists \boldsymbol{x}.\Phi(\boldsymbol{x})$; the class of formulas in *universal disjunctive normal form* ($\forall\mathsf{DNF}$) consists of formulas of the form $\forall \boldsymbol{x}.\Phi(\boldsymbol{x})$, where Φ is a disjunction of conjunctive clauses. The class of

formulas in *conjunctive normal form* (CNF) consists of ∃CNF and ∀CNF formulas. The class of formulas in *disjunctive normal form* (DNF) consists of ∃DNF and ∀DNF formulas. A formula is *positive* if it contains only positive literals. We also write kCNF, or kDNF, to denote the class of formulas, where k denotes the maximal number of atoms that a clause can contain.

2.2 Databases and Query Answering

A *database* $\mathcal{D}$ over a (finite) relational vocabulary σ is a finite set of ground atoms over σ. We follow a model-theoretic approach and view a database as a *Herbrand interpretation*, where the atoms that appear in the database are mapped to *true*, while the ones not in the database are mapped to *false*,

Note that this semantics implies the *closed-world assumption* (CWA) [60], and the *closed-domain assumption* (CDA), i.e., restricts the domain of an interpretation to a *finite*, *fixed* set of constants; namely, to database constants. As a matter of fact, such interpretations presume that the *domain is complete*. Finally, the *unique name assumption* (UNA) ensures a bijection between the database constants and the domain: it is not possible to refer to the same individual in the domain with two different constant names. These simplifying assumptions of databases are useful for a variety of reasons. At the same time, it becomes very easy to produce some undesirable consequences under these assumptions, as we will elaborate. We will revisit some of these assumptions, and discuss their implications; for a recent survey on these assumptions, see e.g. [9].

The most fundamental task in databases is *query answering*; that is, given a database $\mathcal{D}$ and a formula $\Phi(x_1, \ldots, x_n)$ of first-order logic, to decide whether there exists assignments to the free variables $x_1, \ldots, x_n$ such that the resulting formula is satisfied by the database. Importantly, here, the variable assignments are of a special type, also called *substitutions*. Formally, a *substitution* $[x/t]$ replaces all occurrences of the variable x by some database constant t in some formula $\Phi[x, y]$, denoted $\Phi[x/t]$.

Given these, we can now formulate query answering as a decision problem. Let σ be a relational vocabulary; $\Phi(x_1, \ldots, x_n)$ be a first-order formula over σ; and $\mathcal{D}$ be a database over σ. Then, *query answering* is to decide whether $\mathcal{D} \models \Phi[x_1/a_1, \ldots, x_n/a_n]$ for a given substitution (answer) $[x_1/a_1, \ldots, x_n/a_n]$ to the free variables $x_1, \ldots, x_n$. For a Boolean formula Φ, *Boolean query answering* (or simply *query evaluation*) is to decide whether $\mathcal{D} \models \Phi$.

There exists a plethora of query languages in the literature. Classical database query languages range from the well-known *conjunctive queries* to arbitrary first-order queries, which we briefly introduce. A *conjunctive query* over σ is an existentially quantified formula $\exists \boldsymbol{x}.\Phi(\boldsymbol{x}, \boldsymbol{y})$, where $\Phi(\boldsymbol{x}, \boldsymbol{y})$ is a conjunction of atoms over σ. A *Boolean conjunctive query* over σ is a conjunctive query without free variables. A *union of conjunctive queries* is a disjunction of conjunctive queries with the same free variables. A *union of conjunctive queries* is Boolean if it does not contain any free variable. The class of *Boolean unions of conjunctive queries* is denoted UCQ.

We always focus on Boolean queries unless explicitly mentioned otherwise. Conjunctive queries are the most common database queries used in practice; thus, they will also be emphasized in this work. Besides, note that full relational algebra corresponds to the class of first-order formulas. Therefore, we include fragments of the class of first-order formulas as query languages in our analysis. In particular, we study $\exists\mathsf{FO}$, $\forall\mathsf{FO}$, and FO queries as query languages. Besides, we sometimes use different syntactic forms to represent relational queries, such as CNF or DNF.

We also speak of matches for Boolean queries. Let Q be a Boolean query over σ, $\mathcal{D}$ a database over σ, and $\mathbf{V}(Q)$ be the set of variables that occur in Q. A mapping $\varphi : \mathbf{V}(Q) \mapsto \mathbf{C}$ is called a *match for the query* Q in $\mathcal{D}$ if $\mathcal{D} \models \varphi(Q)$. For existentially quantified queries, it is sufficient to find a single match, to satisfy a given Boolean query evaluation. Conversely, for universally quantified queries, all mappings must result in a match in order to satisfy the query.

2.3 Complexity Background

We assume some familiarity with complexity theory and refer the reader to standard textbooks in the literature [53, 66]. We now briefly introduce the complexity classes that are most relevant to the presented results.

FP is the class of functions $f : \{0,1\}^* \to \{0,1\}^*$ computable by a polynomial-time deterministic Turing Machine. The function class #P [71] is central for problems related to counting. The canonical problem for #P is $\#\mathsf{SAT}$, that is, given a propositional formula φ, the task of computing the number of satisfying assignments to φ. In this tutorial, we mostly focus on decision complexity classes. Intuitively, the complexity class PP [32] can be seen as the decision variant of #P. Formally, PP is the set of languages recognized by a polynomial time nondeterministic Turing machine that accepts an input if and only if *more than half* of the computation paths are accepting. The canonical problem for PP is MAJSAT, that is, given a propositional formula φ, the problem of deciding whether the majority of the assignments to φ are satisfying. Importantly, PP is closed under *truth table reductions* [6]; in particular, this implies that PP is closed under *complement*, *union*, and *intersection*. PP contains NP and is contained in PSpace.

Another class of interest is $\mathrm{NP}^{\mathrm{PP}}$, which intuitively combines search and optimization problems. A natural canonical problem for this class is $\mathsf{EMAJSAT}$ [48], that is, given a propositional formula φ and a set of distinguished variables $\boldsymbol{x}$ from φ, is there an assignment μ to $\boldsymbol{x}$-variables such that majority of the assignments τ that extend μ satifies φ. $\mathrm{NP}^{\mathrm{PP}}$ appears as a fundamental class for probabilistic inference and planning tasks. The inclusion relationships between these complexity classes can be summarized as follows:

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{P}^{\mathrm{PP}} = \mathrm{P}^{\#\mathrm{P}} \subseteq \mathrm{NP}^{\mathrm{PP}} \subseteq \mathrm{PSpace} \subseteq \mathrm{Exp},$$

where $\mathrm{P}^{\mathrm{PP}} = \mathrm{P}^{\#\mathrm{P}}$ is due to Toda's result [70].

We also make references to lower complexity classes characterized by Boolean circuits. AC^0 consists of the languages recognized by Boolean circuits with constant depth and polynomial number of *unbounded fan-in* AND and OR gates. TC^0 consists of the languages recognized by Boolean circuits with constant depth and a *polynomial number* of *unbounded fan-in* AND, OR, and MAJORITY gates. Unlike Turing machines, Boolean circuits are non-uniform models of computation; that is, inputs of different size are processed by different circuits. It is therefore common to impose some *uniformity* conditions that require the existence of some resource-bounded Turing machine that, on an input, produces a description of the individual circuit. The most widely accepted uniformity condition for these classes is DLogTime-uniformity, bounding the computation in accordance to a logarithmic-time deterministic Turing machine. For a detailed treatment of the subject, we refer to the relevant literature [73]. The relationships between these classes can be summarized as follows:

$$\mathrm{AC}^0 \subseteq \mathrm{TC}^0 \subseteq \textsc{LogSpace} \subseteq \mathrm{P}$$

When analyzing the computational complexity, we restrict ourselves to the data complexity throughout this tutorial, in order to achieve a more focused presentation. As usual, the *data complexity* is calculated only based on the size of the database, i.e., the query is assumed to be fixed [72].

3 Probabilistic Databases

In this section, we introduce probabilistic databases (PDBs) [68] as one of the most basic data models underlying probabilistic knowledge bases. Our analysis is organized in two subsections; first, we give an overview of *query evaluation* methods in PDBs, and afterwards, we study *maximal posterior computation* tasks, introduced for PDBs [37, 14].

We adopt the simplest probabilistic database model, which is based on the *tuple-independence assumption*. For alternative models, we refer the reader to the rich literature of probabilistic databases; see e.g. [68] and the references therein. Tuple-independent probabilistic databases generalize classical databases by associating every database atom with a probability value.

Definition 1. A *probabilistic database* (PDB) $\mathcal{P}$ for a vocabulary σ is a finite set of *tuples* of the form $\langle t : p \rangle$, where t is a σ-atom and $p \in (0, 1]$. Moreover, if $\langle t : p \rangle \in \mathcal{P}$ and $\langle t : q \rangle \in \mathcal{P}$, then $p = q$.

Table 1 shows a sample PDB $\mathcal{P}_m$, where each row in a table represents an atom that is associated with a probability value. Semantically, a PDB can be viewed as a factored representation of exponentially many *possible worlds* (classical databases), each of which has a probability to be true. Both in the AI [63, 64, 57, 24] and the database literature [68], this is commonly referred to as the *possible worlds semantics*.

In PDBs, each database atom is viewed as an independent random variable by the *tuple-independence assumption*. Each *world* is then simply a *classical*

Table 1. The PDB $\mathcal{P}_m$ represented in terms of database tables, where each row is interpreted as a probabilistic atom.

StarredIn		P
deNiro	taxiDriver	0.7
thurman	pulpFiction	0.1
travolta	pulpFiction	0.3

DirectedBy		P
pulpFiction	tarantino	0.8
taxiDriver	scorsese	0.6
winterSleep	ceylan	0.8

database, which sets a choice for all database atoms in the PDB. Furthermore, the *closed-world assumption* forces all atoms that are not present in the database to have probability zero.

Definition 2. A PDB $\mathcal{P}$ for vocabulary σ induces a *unique probability distribution* $\mathrm{P}_{\mathcal{P}}$ over the set of (possible worlds) $\mathcal{D}$ such that

$$\mathrm{P}_{\mathcal{P}}(\mathcal{D}) = \prod_{t \in \mathcal{D}} \mathrm{P}_{\mathcal{P}}(t) \prod_{t \notin \mathcal{D}} (1 - \mathrm{P}_{\mathcal{P}}(t)),$$

where the probability of each atom is given as

$$\mathrm{P}_{\mathcal{P}}(t) = \begin{cases} p \text{ if } \langle t : p \rangle \in \mathcal{P} \\ 0 \text{ otherwise.} \end{cases}$$

Whenever the probabilistic database is clear from the context, we simply write $\mathrm{P}(t)$, instead of $\mathrm{P}_{\mathcal{P}}(t)$. We say that a database is *induced by* a PDB $\mathcal{P}$ if it is a possible world (with a non-zero probability) of $\mathcal{P}$.

Observe that the choice of setting $\mathrm{P}_{\mathcal{P}}(t) = 0$ for tuples missing from PDB $\mathcal{P}$ is a *probabilistic counterpart* of the closed-world assumption. Let us now illustrate the semantics of PDBs on a simple example.

Example 1. Consider the PDB $\mathcal{P}_m$ given in Table 1. The probability of the world $\mathcal{D}_1$, given as

$$\mathcal{D}_1 \coloneqq \{\mathsf{StarredIn}(\mathsf{deNiro}, \mathsf{taxiDriver}), \mathsf{DirectedBy}(\mathsf{taxiDriver}, \mathsf{scorsese})\},$$

can then be computed by multiplying the probabilities of the atoms that appear in $\mathcal{D}_1$ with the dual probability of the atoms that do not appear in $\mathcal{D}_1$ as follows

$$\mathrm{P}(\mathcal{D}_1) = 0.7 \cdot (1 - 0.1) \cdot (1 - 0.3) \cdot (1 - 0.8) \cdot 0.6 \cdot (1 - 0.8).$$

■

The semantics of queries is given through the possible worlds semantics, which amounts to walking through all the possible worlds and summing over the probabilities of those worlds that satisfy the query.

Definition 3 (query semantics). Let Q be a Boolean query and $\mathcal{P}$ be a PDB. The *probability* of Q in the PDB $\mathcal{P}$ is defined as

$$\mathrm{P}_{\mathcal{P}}(Q) = \sum_{\mathcal{D} \models Q} \mathrm{P}_{\mathcal{P}}(\mathcal{D}),$$

where $\mathcal{D}$ ranges over all possible worlds.

In general, there are exponentially many worlds, and in some cases, it is unavoidable to go through all of them in order to compute the probability. This is computationally intractable, but as we shall see later, in some cases, computing the query probability is actually easy.

Example 2. Consider again the PDB $\mathcal{P}_m$. In order to evaluate the following Boolean query

$$Q \coloneqq \exists x, y \; \mathsf{StarredIn}(x, y) \wedge \mathsf{DirectedBy}(y, \mathsf{scorsese}),$$

on $\mathcal{P}_m$, we can naïvely check, for each world $\mathcal{D}$, whether $\mathcal{D} \models Q$. One such world is $\mathcal{D}_1$, as it clearly satisfies $\mathcal{D}_1 \models Q$. Afterwards, we only need to sum over the probabilities of the worlds, for which the satisfaction relation holds, in order to obtain the probability of the query. ■

In the given example, it is *easy* to compute the probability of the query. Notably, this is the case for any PDB and not only for our toy PDB. The next section is dedicated to give an understanding of *easy* and *hard* queries.

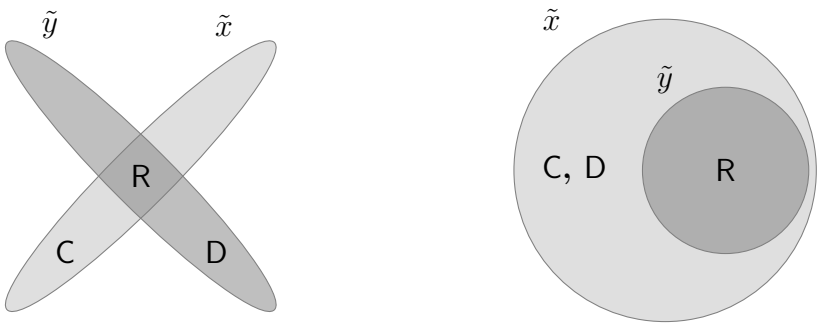
3.1 Query Evaluation in Probabilistic Databases

In this section, we provide a short overview on existing complexity results for inference in (tuple-independent) probabilistic databases including a data complexity dichotomy result. In our analysis, we are interested in the decision problem of probabilistic query evaluation, as defined next.

Definition 4 (probabilistic query evaluation). Given a PDB $\mathcal{P}$, a query Q and a threshold value $p \in [0, 1)$, *probabilistic query evaluation*, denoted PQE, is to decide whether $\mathrm{P}_{\mathcal{P}}(Q) > p$. PQE is parametrized with a particular query language; thus, we write $\mathsf{PQE}(\mathsf{Q})$ to define PQE on the class of Q queries.

The data complexity of query evaluation depends heavily on the structure of the query. In a remarkable result [21], it has been shown that the probability of a UCQ can be computed either in FP or it is #P-hard on any PDB. Using the usual terminology [21], we say that queries are *safe* if the computation problem is in FP, and *unsafe*, otherwise. Probabilistic query evaluation, as defined here, is the corresponding decision problem. It is easy to see that this problem is either in P or it is PP-complete as a corollary to the result of [21].

Corollary 1. $\mathsf{PQE}(\mathsf{UCQ})$ *is either in* P *or it is* PP*-complete for PDBs in data complexity under polynomial-time Turing reductions.*



(a) $Q_{\mathsf{NH}} \coloneqq \exists x, y\, \mathsf{C}(x) \wedge \mathsf{R}(x, y) \wedge \mathsf{D}(y)$ (b) $Q_{\mathsf{H}} \coloneqq \exists x, y\, \mathsf{C}(x) \wedge \mathsf{R}(x, y) \wedge \mathsf{D}(x)$

Fig. 1. Venn diagram for the queries Q_{NH} (non-hierarchical) and Q_{H} (hierarchical).

Proof. Let a UCQ Q be *safe* for PDBs. Then, for any PDB $\mathcal{P}$, the computation problem $\mathrm{P}(Q)$ uses only polynomial time. As a consequence, it is possible to decide whether the probability exceeds a given threshold p in polynomial time. Thus, $\mathsf{PQE}(\mathsf{UCQ})$ is in P for all *safe* UCQ queries.

Conversely, let a UCQ Q be *unsafe* for PDBs. Then, the problem of computing $\mathrm{P}(Q)$ on a given PDB $\mathcal{P}$ is #P-hard *under polynomial-time Turing reductions*. Let us loosely denote by $\mathrm{P}(Q)$ the problem of computing $\mathrm{P}(Q)$. We now only show that PP is contained in $\mathrm{P}^{\mathsf{PQE}(Q)}$, i.e., $\mathsf{PQE}(Q)$ is PP-hard in data complexity under polynomial-time Turing reductions.

To show this, let A be any other problem in PP. By assumption, its computation problem, denoted $\#\mathsf{A}$, is contained in $\mathrm{FP}^{\mathrm{P}(Q)}$, i.e., there is a polynomial-time Turing machine with oracle $\mathrm{P}(Q)$ that computes the output for $\#\mathsf{A}$. We can adapt this Turing machine then to compare the output to some threshold, which means that A is contained in $\mathrm{P}^{\mathrm{P}(Q)}$. We also know that $\mathrm{P}(Q)$ is contained in $\mathrm{FP}^{\mathsf{PQE}(Q)}$, as we can perform a binary search over the interval $[0, 1]$ to compute the precise probability $\mathrm{P}(Q)$. This implies that A is contained in $\mathrm{P}^{\mathfrak{C}}$, where $\mathfrak{C} = \mathrm{FP}^{\mathsf{PQE}(Q)}$. Finally, note that the intermediate oracle does not provide any additional computational power (as this computation can be performed by the polynomial-time Turing machine and the oracle $\mathsf{PQE}(Q)$ can be queried directly). This shows that A is in $\mathrm{P}^{\mathsf{PQE}(Q)}$, which proves the result. □

We use the same terminology also for the associated decision problem: we say that a query Q is *safe* if $\mathsf{PQE}(Q)$ is in P, and *unsafe*, otherwise. Historically, a dichotomy is considered first by [34], and later, the so-called *small dichotomy result* has been proven by [20], which applies to a subclass of conjunctive queries. As it gives nice insights on the larger dichotomy result [21], we look into this result, in more detail. The small dichotomy applies to all conjunctive queries without self-joins, i.e., conjunctive queries with non-repeating relation symbols. It asserts that a self-join free query is hard if and only if it is *nonhierarchical*,

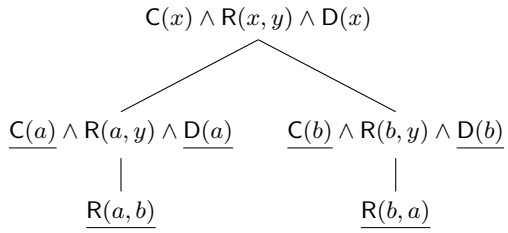


Fig. 2. Decomposition tree of a safe query for the grounding $[x/a, y/b]$ (left branch) and $[x/b, y/a]$ (right branch). Different branches of the tree do not share an atom, which ensures independence.

and it is safe, otherwise. It is therefore crucial to understand *hierarchical* and *nonhierarchical* queries.

Definition 5 (hierarchical queries). Let Q be a conjunctive query. For any variable x that appears in the query Q, its *x-cover*, denoted $\tilde{x}$, is defined as the set of all relation names that have the variable x as an argument. Two covers $\tilde{x}$ and $\tilde{y}$ are *pairwise hierarchical* if and only if $\tilde{x} \cap \tilde{y} \neq \emptyset$ implies $\tilde{x} \subseteq \tilde{y}$ or $\tilde{y} \subseteq \tilde{x}$. A query Q is *hierarchical* if every cover $\tilde{x}, \tilde{y}$ is *pairwise hierarchical*; otherwise, it is called *nonhierarchical*.

Let us consider the query $Q_{\mathsf{NH}} \coloneqq \exists x, y\, \mathsf{C}(x) \wedge \mathsf{R}(x, y) \wedge \mathsf{D}(y)$. It is easy to see that this query is not hierarchical, since the relation R occurs in both covers $\tilde{x}$ and $\tilde{y}$ (as depicted in Figure 1a). This simple join query is already unsafe, as shown in [20].

Theorem 1 (Thm.7,[20]). *Q_{NH} is unsafe.*

Proof. We provide a reduction from the model counting problem, that is, given a propositional formula φ, the problem of computing the model count of φ, denoted $\#\varphi$. Provan and Ball [58] showed that computing $\#\varphi$ is #P-complete (under Turing reductions) even for bipartite monotone 2DNF Boolean formulas φ, i.e., when the propositional variables can be partitioned into $X = x_1, \ldots, x_m$ and $Y = y_1, \ldots, y_n$ such that $\varphi = c_1 \vee \cdots \vee c_l$, where each clause c_i has the form $x_j \wedge y_k$, $x_j \in X, y_k \in Y$.

Given φ, we define the probabilistic database $\mathcal{P}_\varphi$, which contains the atoms $\langle \mathsf{C}(x_1) : 0.5 \rangle$, $\ldots$, $\langle \mathsf{C}(x_m) : 0.5 \rangle$, $\langle \mathsf{D}(y_1) : 0.5 \rangle$, $\ldots$, $\langle \mathsf{D}(y_n) : 0.5 \rangle$, and for every clause $(x_j \wedge y_k)$ an atom $\langle \mathsf{R}(x_j, y_k) : 1 \rangle$. It is then easy to verify that $\#\varphi = \mathrm{P}(Q_{\mathsf{NH}}) \cdot 2^{m+n}$, which concludes the result. □

Similar reductions can be obtained for all (self-join free) conjunctive queries that are non-hierarchical. Note, however, that removing any of the atoms from Q_{NH} results in a safe query. For example, the query $\exists x, y\ \mathsf{C}(x) \wedge \mathsf{R}(x, y)$ is hierarchical and thus safe. The query $Q_{\mathsf{H}} \coloneqq \exists x, y\ \mathsf{C}(x) \wedge \mathsf{R}(x, y) \wedge \mathsf{D}(x)$, as shown in Figure 1b, is yet another example of a safe query.

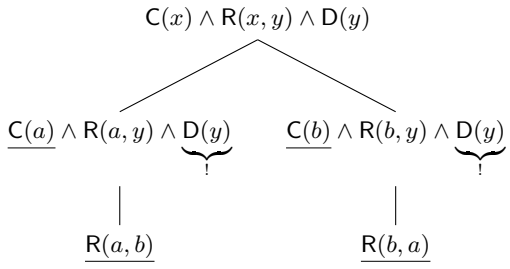


Fig. 3. Decomposition tree of an unsafe query for the grounding $[x/a, y/b]$ (left branch) and $[x/b, y/a]$ (right branch). Different branches of the tree share D-atoms, which makes them dependent.

The intuition behind a safe query is the query being recursively decomposable into sub-queries such that each such sub-query is probabilistically independent. Let us consider the query Q_{H}, as it admits a decomposition, and is safe. We can first ground over x, which results in a query of the form $\exists y\ \mathsf{C}(a) \wedge \mathsf{R}(a, y) \wedge \mathsf{D}(a)$ for a grounding $[x/a]$. The atoms in the resulting query do not share a relation name or a variable, and since we additionally assume tuple independence, it follows that the probability of each atom is independent. Thus, their probabilities can be computed separately and combined afterwards using appropriate rules of probability.

Note that our observation for the independence is also valid for all different groundings of Q_{H}. For example, the groundings $Q_{\mathsf{H}}[x/a]$ and $Q_{\mathsf{H}}[x/b]$, are probabilistically independent, since after applying a grounding over y, we obtain *mutually disjoint sets* of ground atoms. That is, once x is mapped to different constants, then all mappings for y will result in different sets of atoms. As a result, their probabilities can be computed separately and combined afterwards. The decomposition of the safe query Q_{H} is depicted in Figure 2 in terms of a tree. The key ingredient in this example is related to the variable x, which serves as a separator variable in the first place and allows us to further simplify the query.

Definition 6 (separator variable). Let Q be a first-order query. A variable x in Q is a *separator variable* if x appears in all atoms of Q and for any two different atoms of the same relation R, the variable x occurs in the same position.

Note that the query Q_{NH} has no separator variable, since neither x nor y serve as a separator variable. Intuitively, this means that the query cannot be decomposed into independent sub-queries. For example, two different groundings $Q_{\mathsf{NH}}[x/a]$ and $Q_{\mathsf{NH}}[x/b]$ are not independent for Q_{NH}, since they do not necessarily result in mutually exclusive sets of atoms once grounded over y, as shown in Figure 3. The small dichotomy theorem uses other rules of probability theory to further simplify the query, and we illustrate this on an example.

Example 3. Consider the hierarchical query $Q_{\mathsf{H}} \coloneqq \exists x, y\ \mathsf{C}(x) \wedge \mathsf{R}(x, y) \wedge \mathsf{D}(x)$. To compute the probability of Q_{H}, we first apply the decomposition based on

the seperator variable x, which yields

$$\mathrm{P}(Q_{\mathsf{H}}) = 1 - \prod_{c \in \mathbf{C}} \mathrm{P}(\exists y\ \mathsf{C}(c) \wedge \mathsf{R}(c,y) \wedge \mathsf{D}(c)),$$

Here, c ranges over the database constants, and the probability of the resulting expression can be computed by decomposing the conjunctions as

$$\mathrm{P}(\exists y\ \mathsf{C}(c) \wedge \mathsf{R}(c,y) \wedge \mathsf{D}(c)) = \mathrm{P}(\mathsf{C}(c)) \cdot \mathrm{P}(\exists y\ \mathsf{R}(c,y)) \cdot \mathrm{P}(\mathsf{D}(c)).$$

The probabilities of the ground atoms $\mathsf{C}(c)$, $\mathsf{D}(c)$ can be read off from the given probabilistic database; thus, it only remains to apply the grounding in $\mathsf{R}(c,y)$, which results in

$$\mathrm{P}(\exists y\ \mathsf{R}(c,y)) = 1 - \prod_{d \in \mathbf{C}} \mathrm{P}(\mathsf{R}(c,d)).$$

■

The dichotomy for unions of conjunctive queries is much more intricate and a characterization of safe queries is unfortunately not easy. Thus, an algorithm is given in [21] to compute the probability of all safe queries by recursively applying the simplification rules on the query. This algorithm is complete, i.e., when the algorithm fails on the query, then the query is unsafe. Later, a lifted inference algorithm, called $\mathsf{Lift}^{\mathsf{R}}$, was proposed in [38], which was also proven to be complete. Afterwards, this algorithm has also been extended to an open-world semantics in [15], where it has also been noted that this algorithm runs in linear time for PDBs in the size of the database, under reasonable assumptions, such as unit arithmetic cost assumption for all arithmetic operations. For full details on different algorithms and their properties, we refer to the relevant literature [21, 38, 15, 13].

3.2 Maximal Posterior Computations for Probabilistic Databases

Forming the foundations of large-scale knowledge bases, probabilistic databases have been widely studied in the literature. In particular, probabilistic query evaluation has been investigated intensively as a central inference mechanism. However, despite its power, query evaluation alone cannot extract all the relevant information encompassed in large-scale knowledge bases.

To exploit this potential, two additional inference tasks are proposed [37, 14], namely, finding the *most probable database* and the *most probable hypothesis* for a given query, both of which are inspired by *maximal posterior probability* computations in probabilistic graphical models (PGMs) [44]. Let us briefly explain why probabilistic query evaluation alone may not be sufficient on the following example.

Example 4. Consider the PDB $\mathcal{P}_v$ given in Table 2 and the conjunctive query

$$Q_{\mathsf{fr}} = \exists x, y\, Q_{\mathsf{veg}}(x) \wedge \mathsf{FriendOf}(x,y) \wedge Q_{\mathsf{veg}}(y).$$

Table 2. The probabilistic database $\mathcal{P}_v$ encodes dietary regimes of some individuals, and the friendship relation among those individuals.

Vegetarian	
alice	0.7
bob	0.9
chris	0.6

FriendOf		
alice	bob	0.7
alice	chris	0.8
bob	chris	0.1

Eats		
bob	spinach	0.7
chris	mussels	0.8
alice	broccoli	0.2

Meat	
shrimp	0.7
mussels	0.9
seahorse	0.3

In the given PDB, alice, bob, and chris are all vegetarians and friends with each other with some probability. We can now ask the probability of vegetarians being friends with vegetarians. The query

$$Q_{\mathsf{fr}}[x/\mathsf{bob}] = \exists y\ Q_{\mathsf{veg}}(\mathsf{bob}) \wedge \mathsf{FriendOf}(\mathsf{bob}, y) \wedge Q_{\mathsf{veg}}(y)$$

is a special case of Q_{fr}, which asks whether bob has vegetarian friends. Its probability can be computed as $\mathrm{P}(Q_{\mathsf{fr}}[x/\mathsf{bob}]) = 0.9 \cdot 0.1 \cdot 0.6 = 0.054$.

Suppose now that we observe that $Q_{\mathsf{fr}}[x/\mathsf{bob}]$ is true and would like to learn what best explains this observation relative to the underlying probabilistic database. To be able to explain such an observation, we need different inference tasks than probabilistic query answering. ■

The *most probable database* problem (analogous to *most probable explanations (MPE)* in PGMs), first proposed in [37], is the problem of determining the (classical) database with the largest probability that satisfies a given query. Intuitively, the query defines constraints on the data, and the goal is to find the most probable database that satisfies these constraints. The *most probable hypothesis* problem (analogous to *maximum a posteriori problems (MAP)* in PGMs), first proposed in [14], only asks for *partial* databases satisfying the query. The most probable hypothesis contains only atoms that contribute to the satisfaction condition of the query, which allows to more precisely pinpoint the most likely explanations of the query.

The Most Probable Database Problem. The need for alternative inference mechanisms for probabilistic knowledge bases has been observed before, and the *most probable database* problem has been proposed in [37] as follows.

Definition 7. Let $\mathcal{P}$ be a probabilistic database and Q a query. The *most probable database* for Q over $\mathcal{P}$ is given by

$$\arg\max_{\mathcal{D} \models Q} \mathrm{P}(\mathcal{D}),$$

where $\mathcal{D}$ ranges over all worlds induced by $\mathcal{P}$.

Intuitively, a probabilistic database defines a probability distribution over exponentially many classical databases, and the most probable database is the element in this collection that has the highest probability, while still satisfying the

Table 3. The most probable database for the query Q_{fr} over the PDB $\mathcal{P}_v$.

Vegetarian		FriendOf		Eats		Meat	
alice	0.7	alice bob	0.7	bob spinach	0.7	shrimp	0.7
bob	0.9	alice chris	0.8	chris mussels	0.8	mussels	0.9
chris	0.6	bob chris	0.1	alice broccoli	0.2	seahorse	0.3

given query. This can be seen as the best instantiation of a probabilistic model, and hence analogous to most probable explanation in Bayesian networks [23]. We illustrate the most probable database on our running example.

Example 5. Consider the given PDB $\mathcal{P}_v$. We can now filter the most probable database that satisfies the query

$$Q_{\mathsf{fr}} = \exists x, y \; Q_{\mathsf{veg}}(x) \wedge \mathsf{FriendOf}(x, y) \wedge Q_{\mathsf{veg}}(y).$$

Intuitively, the atoms $\mathsf{Vegetarian}(\mathsf{alice})$, $\mathsf{Vegetarian}(\mathsf{bob})$, and $\mathsf{FriendOf}(\mathsf{alice}, \mathsf{bob})$ need to be included in the most probable database, as they satisfy the query with the highest probability (compared to other possible matches). Since the query is monotone, for the remaining atoms, we only need to decide whether including them results in a higher probability than excluding them, which amounts to deciding whether their probability is greater than 0.5. Table 3 shows the most probable database for Q_{fr} over the PDB $\mathcal{P}_v$, where all atoms that belong to the most probable database are highlighted. ■

Identifying the most probable database in this example is rather straightforward, and as we shall see later, this is tightly related to the query language that is considered. We now illustrate that identifying the most probable database can be more cumbersome if we consider other query languages; in particular, $\forall\mathsf{FO}$ queries.

Example 6. Consider again the PDB $\mathcal{P}_v$ and the query

$$Q_{\mathsf{veg}} = \forall x, y \; \neg\mathsf{Vegetarian}(x) \vee \neg\mathsf{Eats}(x, y) \vee \neg\mathsf{Meat}(y),$$

which defines the constraints of being a vegetarian, which is violated by the atoms $\mathsf{Vegetarian}(\mathsf{chris})$, $\mathsf{Eats}(\mathsf{chris}, \mathsf{mussels})$, and $\mathsf{Meat}(\mathsf{mussels})$. Therefore, the most probable database for Q_{veg} cannot contain all three of them, i.e., one of them has to be removed (from the explanation). In this case, it is easy to see that $\mathsf{Vegetarian}(\mathsf{chris})$ needs to be removed, as it has the lowest probability among them. Thus, the most probable database (in this case unique) contains all atoms of $\mathcal{P}_v$ that have a probability above 0.5, except for $\mathsf{Vegetarian}(\mathsf{chris})$.

Suppose now that we have observed $Q_{\mathsf{fr}}[x/\mathsf{bob}]$, and we are interested in finding an explanation for this observation under the constraint of Q_{veg}, which is specified by the query

$$Q_{\mathsf{vf}} = Q_{\mathsf{fr}}[x/\mathsf{bob}] \wedge Q_{\mathsf{veg}}.$$

Table 4. The most probable database for the query Q_{vf} over the PDB $\mathcal{P}_v$.

Vegetarian	
alice	0.7
bob	0.9
chris	0.6

FriendOf	
alice bob	0.7
alice chris	0.8
bob chris	0.1

Eats	
bob spinach	0.7
chris mussels	0.8
alice broccoli	0.2

Meat	
shrimp	0.7
mussels	0.9
seahorse	0.3

Observe that $\mathsf{Vegetarian}(\mathsf{chris})$ and $\mathsf{FriendOf}(\mathsf{bob}, \mathsf{chris})$ must be in the explanation to satisfy $Q_{\mathsf{fr}}[x/\mathsf{bob}]$. Moreover, either $\mathsf{Eats}(\mathsf{chris}, \mathsf{mussels})$ or $\mathsf{Meat}(\mathsf{mussels})$ has to be excluded from the most probable database, since otherwise Q_{veg} will be violated. The resulting most probable database is highlighted in Table 4. ■

The most probable database is formulated as a decision problem as follows.

Definition 8 (MPD). Let Q be a query, $\mathcal{P}$ a probabilistic database, and $p \in (0, 1]$ a threshold. MPD is the problem of deciding whether there exists a database $\mathcal{D}$ that satisfies Q with $\mathrm{P}(\mathcal{D}) > p$. MPD is parametrized with a particular query language; thus, we write MPD(Q) to define MPD on the class Q of queries.

The first result that we present concerns the well-known class of *unions of conjunctive queries*: MPD can be solved using at most *logarithmic space* and *polynomial time*. More precisely, it is possible to encode the MPD problem uniformly into a class of TC^0 circuits.

Theorem 2. MPD(UCQ) *is in* DLogTime*-uniform* TC^0 *in data complexity.*

Proof. Let $\mathcal{P}$ be a PDB, Q a UCQ, and $p \in [0, 1)$ a threshold value. In principle, we can enumerate all the databases induced by the PDB $\mathcal{P}$ and decide whether there is a database that satisfies Q and has a probability greater or equal than p. This would require exponential time, as there are potentially exponentially many databases (worlds) induced by a probabilistic database. Fortunately, this can be avoided, as the satisfaction relation for unions of conjunctive queries is monotone: once a database satisfies a UCQ, then any superset of this database will satisfy the UCQ. We can, therefore, design an algorithm, which initiates the database with the atoms resulting from a match for the query (where the match is over the atoms that appear with a positive probability in the PDB $\mathcal{P}$) and extends this to a database with maximal probability, as follows: it adds only those atoms from the PDB $\mathcal{P}$ to the database that appear with a probability higher than 0.5, ensuring to obtain the database with the maximal probability. As a consequence, all these databases satisfy Q and if, furthermore, there is a database $\mathcal{D}$ among them with $\mathrm{P}(\mathcal{D}) \geq p$, then the algorithm answers yes; otherwise, it answers no.

It is easy to see that this algorithm is correct. However, if performed naïvely, as described, it results in a polynomial blow-up: there are polynomially many matches for the query (in data complexity), and for each match, constructing a database requires polynomial time and space. Observe, on the other hand, that we can uniformly access every match through an AC^0 circuit, and we can

avoid explicitly constructing a database for each such match. More concretely, since we are only interested in the probability of the resulting database, we can perform an iterated multiplication over the probabilities of the individual atoms: if $\langle a : q \rangle \in \mathcal{P}$, where $q > 0.5$, we multiply with q, otherwise, we multiply with $1-q$. We can perform such iterated multiplication with TC^0 circuits that can be constructed in DLogTime [39]. Finally, it is sufficient to compare the resulting number with the threshold value p. This last comparison operation can also be done using uniform AC^0 circuits. This puts $\mathsf{MPD}(\mathsf{UCQ})$ in DLogTime-uniform TC^0 in data complexity. □

This low data complexity result triggers an obvious question: what are the sources of tractability? The answer is partially hidden behind the fact that the satisfaction relation for unions of conjunctive queries is monotone. This allows us to reduce the search space to polynomially many databases, which are obtained from different matches of the query, in a unique way. Is the monotonicity a strict requirement for tractability? It turns out that the TC^0 upper bound can be strengthened towards all *existential queries*, i.e., existentially quantified formulas that allow negations in front of query atoms.

Theorem 3. $\mathsf{MPD}(\exists\mathsf{FO})$ *is in* DLogTime*-uniform* TC^0 *in data complexity.*

Proof. Let $\mathcal{P}$ be a PDB, Q a $\exists\mathsf{FO}$ query, and $p \in [0, 1)$ a threshold value. Analogously to the proof of Theorem 2, we can design an algorithm that initiates the database with the positive atoms from a match, while now also keeping the record of the negative atoms from this match. We can again enumerate all the matches, and for each such match, perform an iterated multiplication over the probabilities of the atoms. The only difference is that now we need to exclude the atoms from the database that appear negatively in the match to ensure the satisfaction of the query. Thus, for all probabilistic atoms $\langle a : q \rangle \in \mathcal{P}$, we check whether $\neg a$ appears in the match, and if so, then we multiply with $1 - q$; otherwise, we check whether $q > 0.5$ and multiply with q, if this test is positive, and multiply with $1 - q$, if this test is negative. It is then sufficient to compare the result of each multiplication with the threshold value p: if one of the resulting multiplications is greater than or equal to p, then the algorithm answers yes, otherwise, it answers no. It is easy to verify the correctness of this algorithm.

By similar arguments as in the proof of Theorem 2, we can check the matches in AC^0, perform the respective multiplications in TC^0 and the respective comparisons in AC^0. Thus, $\mathsf{MPD}(\exists\mathsf{FO})$ is in DLogTime-uniform TC^0 in data complexity. □

In some sense, the presented tractability result implies that nonmonotonicity is not harmful if we restrict our attention to existential queries. This is because the nonmonotonicity involved here is of a limited type and does not lead to a combinatorial blow-up. This picture changes once we focus on universally quantified queries: nonmonotonicity combined with universal quantification creates nondeterministic choices, which we use to prove an NP-hardness result for the most probable database problem. We note that this result is very similar to the

hardness result obtained in [37], only on a different query. Additionally, we show that the complexity bound is tight even if we consider FO queries.

Theorem 4. $\mathsf{MPD}(\forall\mathsf{FO})$ *is* NP*-complete in data complexity, and so is* $\mathsf{MPD}(\mathsf{FO})$.

Proof. We first show that $\mathsf{MPD}(\mathsf{FO})$ can be solved in NP. Let $\mathcal{P}$ be a PDB, Q a FO query, and $p \in [0, 1)$ a threshold value. To solve the decision problem, we first guess a world $\mathcal{D}$, and then verify both that the query is satisfied, i.e., $\mathcal{D} \models Q$, and that the threshold is met, i.e., $\mathrm{P}(\mathcal{D}) \geq p$. Note that all these computations can be done using a nondeterministic Turing machine, since only the first step is nondeterministic, and both of the verification steps can be done in polynomial time in data complexity.

To prove hardness, we provide a reduction from the satisfiability of propositional 3CNF formulas. Let $\varphi = \bigwedge_i \varphi_i$ be a propositional formula in 3CNF. We define the $\forall$FO query

$$\begin{aligned} Q_{\mathsf{SAT}} := \forall x, y, z \, (\quad & \mathsf{L}(x) \vee \quad \mathsf{L}(y) \vee \quad \mathsf{L}(z) \vee \mathsf{R}_1(x, y, z)) \wedge \\ & (\neg\mathsf{L}(x) \vee \quad \mathsf{L}(y) \vee \quad \mathsf{L}(z) \vee \mathsf{R}_2(x, y, z)) \wedge \\ & (\neg\mathsf{L}(x) \vee \neg\mathsf{L}(y) \vee \quad \mathsf{L}(z) \vee \mathsf{R}_3(x, y, z)) \wedge \\ & (\neg\mathsf{L}(x) \vee \neg\mathsf{L}(y) \vee \neg\mathsf{L}(z) \vee \mathsf{R}_4(x, y, z)) \, , \end{aligned}$$

which is later used to encode the satisfaction conditions of φ. Without loss of generality, we denote with $u_1, \ldots, u_n$ the propositional variables that appear in φ.

We then define the PDB $\mathcal{P}_\varphi$, depending on φ, as follows. For each propositional variable u_j, we add the probabilistic atom $\langle \mathsf{L}(u_j) : 0.5 \rangle$ to the PDB $\mathcal{P}_\varphi$. The clauses φ_j are described with the help of the predicates R_1, …, R_4, each of which corresponds to one type of clause. For example, if we have the clause $\varphi_i = x_1 \vee \neg x_2 \vee \neg x_4$, we add the atom $\langle \mathsf{R}_3(x_4, x_2, x_1) : 0 \rangle$ to $\mathcal{P}_\Phi$, which enforces via Q_{SAT} that either $\neg\mathsf{L}(x_4)$, or $\neg\mathsf{L}(x_2)$, or $\mathsf{L}(x_1)$ holds. All other R-atoms that do not correspond in such a way to one of the clauses are added with probability 1 to $\mathcal{P}_\Phi$.

The construction provided for Q_{SAT} and $\mathcal{P}_\varphi$ is clearly polynomial. Furthermore, the query is fixed, and only $\mathcal{P}_\varphi$ depends on φ. We now show that MPD can be used to answer the satisfiability problem of φ, using this construction.

Claim. The 3CNF formula φ is satisfiable if and only if there exists a database $\mathcal{D}$ induced by $\mathcal{P}_\varphi$ such that $\mathrm{P}(\mathcal{D}) \geq (0.5)^n$ and $\mathcal{D} \models Q_{\mathsf{SAT}}$ (where n is the number of variables appearing in φ).

To prove the claim, suppose that φ is satisfiable, and let μ be such a satisfying assignment. We define a world $\mathcal{D}$ such that it contains all the atoms of the form $\mathsf{L}(u_j)$ if and only if $\mu(u_j) \mapsto 1$ in the given assignment. Moreover, $\mathcal{D}$ contains all the atoms that are assigned the probability 1 in $\mathcal{P}_\varphi$. It is easy to see that $\mathcal{D}$ is one of the worlds induced by $\mathcal{P}_\varphi$. Observe further that $\mathcal{P}_\varphi$ contains n nondeterministic atoms, each with 0.5 probability. By this argument, the probability

of $\mathcal{D}$ is clearly $(0.5)^n$. It only remains to show that $\mathcal{D} \models Q_{\mathsf{SAT}}$, which is easy to verify.

For the other direction, let $\mathcal{D} \models Q_{\mathsf{SAT}}$ and $\mathrm{P}(\mathcal{D}) \geq (0.5)^n$ for some world $\mathcal{D}$. We define an assignment μ by setting the truth value of u_j to 1, if $\mathsf{L}(u_j) \in \mathcal{D}$, and to 0, otherwise. Every world contains exactly one assignment for every variable, by our construction. Thus, the assignment μ is well-defined. It is easy to verify that $\mu \models \varphi$. □

This concludes our analysis for the most probable database problem in the context of PDBs, and this problem is revisited later for ontology-mediated queries.

The Most Probable Hypothesis Problem. The most probable database identifies the most likely state of a probabilistic database relative to a query. However, it has certain limitations, which are analogous to the limitations of finding the *most probable explanations* in PGMs [44]. Most importantly, one is always forced to choose a *complete* database although the query usually affects only a subset of the atoms. That is, it is usually not the case that the whole database is responsible for the goal query to be satisfied. To be able to more precisely pinpoint the explanations of a query, the most probable hypothesis is introduced [14].

Definition 9. The *most probable hypothesis* for a query Q over a PDB $\mathcal{P}$ is

$$\arg\max_{\mathcal{H} \models Q} \sum_{\mathcal{D} \models \mathcal{H}} \mathrm{P}(\mathcal{D}),$$

where $\mathcal{H}$ ranges over sets of atoms t and negated atoms $\neg t$ such that t occurs in $\mathcal{P}$, and $\mathcal{H} \models Q$ holds if and only if all worlds induced by $\mathcal{P}$ that satisfy $\mathcal{H}$ also satisfy Q.

Intuitively, the most probable hypothesis contains atoms only if they contribute to the satisfaction of the query, that is, the most probable hypothesis is a partial explanation. It is still the case that an explanation has to satisfy the query, but to do so, it does not need to make a decision for all the database atoms. Indeed, it is possible to specify some positive and negative atoms that ensure the satisfaction of the query, regardless of the truth value of the remaining database atoms.

Conversely, any database $\mathcal{D}$ with $\mathcal{D} \models \mathcal{H}$ must satisfy Q, and thus the most probable database can be seen as a special case of the most probable hypothesis that has to contain all atoms from a PDB (positively or negatively). We denote the sum inside the maximization, i.e., the probability of the explanation by $\mathrm{P}(\mathcal{H})$. Differently from PGMs, the probability of the explanation can be computed by simply taking the product of the probabilities of the (negated) atoms in $\mathcal{H}$. This is a consequence of the independence assumption and influences the complexity results, as we elaborate later.

Table 5. The most probable hypothesis for the query Q_{vf} over $\mathcal{P}_v$.

Vegetarian		FriendOf		Eats		Meat	
alice	0.7	alice bob	0.7	bob spinach	0.7	shrimp	0.7
bob	0.9	alice chris	0.8	chris mussels	0.8	mussels	0.9
chris	0.6	bob chris	0.1	alice broccoli	0.2	seahorse	0.3

Example 7. Consider again our running example with the PDB $\mathcal{P}_v$ and recall the query $Q_{\mathsf{vf}} \coloneqq Q_{\mathsf{veg}} \wedge Q_{\mathsf{fr}}[x/\mathsf{bob}]$, where

$$Q_{\mathsf{veg}} \coloneqq \forall x, y\ \neg\mathsf{Vegetarian}(x) \vee \neg\mathsf{Eats}(x, y) \vee \neg\mathsf{Meat}(y),$$
$$Q_{\mathsf{fr}}[x/\mathsf{bob}] \coloneqq \exists y\ Q_{\mathsf{veg}}(\mathsf{bob}) \wedge \mathsf{FriendOf}(\mathsf{bob}, y) \wedge Q_{\mathsf{veg}}(y).$$

Recall that the most probable database for Q_{vf} contains many redundant atoms. The most probable hypothesis $\mathcal{H}$ for the query Q_{vf} contains only 4 atoms, as given in Table 5: as before, light gray highlighting denotes positive atoms, while the dark gray one denotes negated atoms, i.e., $\neg\mathsf{Eats}(\mathsf{chris}, \mathsf{mussels})$. Note that all of these atoms directly influence the satisfaction of the query and thus are part of the explanation. Since the most probable hypothesis contains less atoms, it is more informative than the most probable database. We can compute the probability of the hypothesis by $\mathrm{P}(\mathcal{H}) = 0.9 \cdot 0.6 \cdot 0.1 \cdot (1 - 0.8) = 0.0108$. ■

Whereas the most probable database represents full knowledge about all facts, which corresponds to the common closed-world assumption for (probabilistic) databases, the most probable hypothesis may leave atoms of $\mathcal{P}$ unresolved, which can be seen as a kind of open-world assumption (although the atoms that do not occur in $\mathcal{P}$ are still false). MPH is defined as a decision problem as follows.

Definition 10 (MPH). Let Q be a query, $\mathcal{P}$ a PDB, and $p \in (0, 1]$ a threshold. MPH is the problem of deciding whether there exists a hypothesis $\mathcal{H}$ that satisfies Q with $\mathrm{P}(\mathcal{H}) \geq p$. MPH is parametrized with a particular query language; thus, we write MPH(Q) to define MPH on the class Q of queries.

We again start our analysis with *unions of conjunctive queries* and show that, as is the case for MPD, MPH can also be solved using at most *logarithmic space* and *polynomial time*.

Theorem 5. MPH(UCQ) *is in* DLogTime*-uniform* TC^0 *in data complexity.*

Proof. Let $\mathcal{P}$ be a PDB, Q a UCQ, and $p \in [0, 1)$ a threshold value. It has already been observed that the satisfaction relation for unions of conjunctive queries is monotone, i.e., the fact that once a database satisfies a UCQ, then any superset of this database satisfies the UCQ. Clearly, this also applies to partial explanations: the databases extending the hypothesis $\mathcal{H}$ satisfy the query only if $\mathcal{H}$ (extended with all atoms that have probability 1) is already a match for the query. This means that the hypothesis must be a subset of a ground instance of one of the disjuncts of the UCQ Q. The major difference from MPD is that, once

the explanation is found, we do not need to consider the other database atoms in the PDB. As before, there are only polynomially many such hypotheses in the data complexity, and they can be encoded uniformly into AC^0 circuits, as in the proof of Theorem 2. Moreover, their probabilities can be computed in TC^0 and the comparison with the threshold p can be done again in AC^0. This puts $\mathsf{MPH}(\mathsf{UCQ})$ in DLogTime-uniform TC^0 in data complexity. □

Recall that, for MPD, it was possible to generalize the data tractability result from unions of conjunctive queries to existential queries. It is therefore interesting to know whether the same holds for MPH. Does MPH remain tractable if we consider existential queries? The answer is unfortunately negative: to be able to verify a test such as $\mathcal{H} \models Q$, we need to make sure that all extensions $\mathcal{D}$ of $\mathcal{H}$ satisfy the query and this test is hard, once we allow negations in front of query atoms. We illustrate the effect of negations on a simple example.

Example 8. Consider the following $\exists\mathsf{FO}$ query

$$Q \coloneqq \exists x, y\ (\mathsf{A}(x) \wedge \neg\mathsf{B}(x,y)) \vee (\neg\mathsf{A}(x) \wedge \neg\mathsf{B}(x,y)).$$

To decide MPH on an arbitrary PDB, we could walk through all (partial) matches for the query and then verify $\mathcal{H} \models Q$. However, observe that this verification is not in polynomial time, in general, as there are interactions between the query atoms and these interactions need to be captured by the explanation itself. For instance, the A-atoms in Q are actually redundant, and it is enough to find an explanation that satisfies $\exists x, y\ \neg\mathsf{B}(x,y)$ for some mapping. ■

The next result is for existential queries, and via a reduction from the validity problem of 3DNF formulas, it shows that MPH is coNP-hard for these queries. It is easy to see that this is also a matching upper bound.

Theorem 6. $\mathsf{MPH}(\exists\mathsf{FO})$ *is* coNP*-complete in data complexity.*

Proof. Let $\mathcal{P}$ be a PDB, Q an $\exists\mathsf{FO}$ query, and $p \in [0,1)$ a threshold value. As for membership, consider a nondeterministic Turing machine, which enumerates all partial matches forming the hypothesis $\mathcal{H}$ and answers yes if and only if $\mathrm{P}(\mathcal{H}) \geq p$ and there is *no* database $\mathcal{D}$ that satisfies $\mathcal{D} \models \mathcal{H}$ while $\mathcal{D} \not\models Q$.

To prove hardness, we provide a reduction from the validity of propositional 3DNF formulas. Let $\varphi = \bigvee_i \varphi_i$ be a propositional formula in 3DNF. We first define the following $\exists\mathsf{FO}$ query

$$\begin{aligned} Q_{\mathsf{VAL}} \coloneqq \exists x, y, z\ (\ & \mathsf{L}(x) \wedge \mathsf{L}(y) \wedge \mathsf{L}(z) \wedge \mathsf{R}_1(x,y,z)) \vee \\ & (\neg\mathsf{L}(x) \wedge \mathsf{L}(y) \wedge \mathsf{L}(z) \wedge \mathsf{R}_2(x,y,z)) \vee \\ & (\neg\mathsf{L}(x) \wedge \neg\mathsf{L}(y) \wedge \mathsf{L}(z) \wedge \mathsf{R}_3(x,y,z)) \vee \\ & (\neg\mathsf{L}(x) \wedge \neg\mathsf{L}(y) \wedge \neg\mathsf{L}(z) \wedge \mathsf{R}_4(x,y,z))\,, \end{aligned}$$

which is later used to encode the validity conditions of φ. Without loss of generality, let us denote with $u_1, \ldots, u_n$ the propositional variables that appear in φ. We then define the PDB $\mathcal{P}_\varphi$, depending on φ, as follows.

- For each propositional variable u_j, we add the probabilistic atom $\langle \mathsf{L}(u_j) : 0.5 \rangle$ to the PDB $\mathcal{P}_\varphi$.
- The conjuncts φ_j are described with the help of the predicates R_1, ..., R_4, each of which corresponds to one type of conjunct. For example, if we have a clause $\varphi_j = u_1 \wedge \neg u_2 \wedge \neg u_4$, we add the atom $\langle \mathsf{R}_3(u_4, u_2, u_1) : 1 \rangle$ to $\mathcal{P}_\Phi$, which enforces via Q_{VAL} that the conjunct that includes all atoms $\neg\mathsf{L}(u_4)$, $\neg\mathsf{L}(u_2)$, and $\mathsf{L}(u_1)$ can be true. All other atoms $\mathsf{R}_i(u_k, u_l, u_m)$ that do not correspond in such a way to one of the clauses are added with probability 0 to $\mathcal{P}_\Phi$.

The construction provided for Q_{VAL} and $\mathcal{P}_\varphi$ is clearly polynomial. Furthermore, the query is fixed, and only $\mathcal{P}_\varphi$ depends on φ. We now show that MPH can be used to answer the validity problem of φ, using this construction.

Claim. The 3DNF formula φ is valid if and only if there exists a hypothesis $\mathcal{H}$ over $\mathcal{P}_\Phi$ such that $\mathrm{P}(\mathcal{H}) \geq 1$ and $\mathcal{H} \models Q_{\mathsf{VAL}}$.

To prove the claim, suppose that φ is valid. We show that the empty hypothesis $\mathcal{H} = \emptyset$ satisfies both $\mathrm{P}(\mathcal{H}) \geq 1$ and $\mathcal{H} \models Q_{\mathsf{VAL}}$. It is easy to see that $\mathrm{P}(\mathcal{H}) = 1$ in $\mathcal{P}_\varphi$, as it encodes all possible databases, i.e., worlds. Let us assume by contradiction that there exists a database $\mathcal{D}$ that extends the hypothesis, $\mathcal{H} \models \mathcal{D}$, but does not satisfy the query, i.e., $\mathcal{D} \not\models Q_{\mathsf{VAL}}$. Then, we can use this database to define a valuation for the given propositional formula: define an assignment μ by setting the truth value of u_j to 1, if $\mathsf{L}(u_j) \in \mathcal{D}$, and to 0, otherwise. Every world contains exactly one assignment for every variable, by our construction. Thus, the assignment μ is well-defined. But then it is easy to see that this implies $\mu \not\models \varphi$, which contradicts the validity of φ.

For the other direction, let $\mathcal{H}$ be a hypothesis such that $\mathrm{P}(\mathcal{H}) \geq 1$ and $\mathcal{H} \models Q_{\mathsf{VAL}}$. This can only be the case if the $\mathcal{H}$ is the empty set (as otherwise $\mathrm{P}(\mathcal{H}) \leq 0.5$). This means that any database $\mathcal{D}$ induced by $\mathcal{P}_\varphi$ must satisfy the query. It is easy to see that every database is in one-to-one correspondence with a propositional assignment; thus, we conclude the validity of Q_{VAL}. □

Having shown that MPH is harder than MPD for existential queries, one may wonder whether this is also the case for universal queries. We now show that MPH has the same complexity as MPD for universal queries.

Theorem 7. MPH(∀FO) *is* NP*-complete in data complexity.*

Proof. NP-hardness can be obtained analogously to the proof of Theorem 4, and we leave the hardness proof as an exercise, and focus on the upper bound which is not as straight-forward. Let $\mathcal{P}$ be a PDB, Q a ∀FO query, and $p \in [0, 1)$ a threshold value. To show membership, we nondeterministically guess a hypothesis $\mathcal{H}$ such that the satisfaction relation $\mathcal{H} \models Q$ is ensured. To do so, assume without loss of generality that the universal query is of the form

$$Q \coloneqq \forall x_1, \ldots, x_n \bigwedge_i q_i(x_1, \ldots, x_n),$$

for some finite numbers $i, n > 0$, where each $q_i(x_1, \ldots, x_n)$ is a clause over $x_1, \ldots, x_n$. Our goal is to find a hypothesis that satisfies this query and that meets the threshold value p. To satisfy a universal query, we need to identify all database atoms that could possibly invalidate the query and rule them out. Thus, we consider the negation of the given query

$$\neg Q = \exists x_1, \ldots, x_n \bigvee_i \neg q_i(x_1, \ldots, x_n),$$

which encodes all database atoms that could possibly invalidate the original query Q. Furthermore, to make the effect of the database atoms more concrete, we consider all possible groundings of this query. Let us denote by

$$\neg Q[x_1/a_1, \ldots x_n/a_n]$$

a grounding with database constants a_i. There are polynomially many such groundings in data complexity. We need to ensure that none of the clauses in any of the groundings is satisfied by the hypothesis. Note that this can be achieved by including, for every clause, an atom into the hypothesis that contradicts the respective clause. For example, suppose that a grounding of the query is

$$(\mathsf{A}(a) \wedge \neg \mathsf{B}(b)) \vee \ldots \vee (\neg \mathsf{C}(c) \wedge \neg \mathsf{D}(d)).$$

Then, for the first clause, we have to either add the atom $\neg \mathsf{A}(a)$ or the atom $\mathsf{B}(b)$ to the hypothesis, and similarly for the last clause. The subtlety is that we cannot deterministically decide which atoms to include into the hypothesis (as there are interactions across clauses as well as across different groundings). Therefore, we nondeterministically guess each such choice. In essence, while constructing the hypothesis, we rule out everything that could invalidate the original query. As a consequence, we ensure that $\mathcal{H} \models Q$. It only remains to check whether $\mathrm{P}(\mathcal{H}) \geq p$, which can be done in polynomial time. Thus, we obtain an NP upper bound in data complexity. □

MPH is coNP-complete for ∃FO queries (by Theorem 6) and NP-complete for ∀FO queries (by Theorem 7). By considering a particular query, which combines the power of existential and universal queries, it becomes possible to show Σ_2^{P}-hardness for MPH.

Theorem 8. MPH(FO) *is* Σ_2^{P}*-complete in data complexity.*

Proof. Let $\mathcal{P}$ be a PDB, Q a FO query, and $p \in [0, 1)$ a threshold value. Consider a nondeterministic Turing machine with a (co)NP oracle: given a PDB $\mathcal{P}$, a first-order query Q, and a threshold $p \in (0, 1]$, we can decide whether there exists a hypothesis $\mathcal{H}$ such that $\mathrm{P}(\mathcal{H}) \geq p$ by first guessing a hypothesis $\mathcal{H}$, and verifying whether (i) $\mathrm{P}(\mathcal{H}) \geq p$ and (ii) for *all* databases $\mathcal{D}$ that extend $\mathcal{H}$ and are induced by $\mathcal{P}$, it holds that $\mathcal{D} \models Q$. Verification of (i) can be done in deterministic polynomial time, and (ii) can be done in coNP (the complement is equivalent to the existence of an extension $\mathcal{D}$ and a valuation for the query variables that falsifies Q). This shows that MPH(FO) is in Σ_2^{P} in data complexity.

As for hardness, we provide a reduction from validity of quantified Boolean formulas of the form $\Phi = \exists u_1, \ldots, u_n \forall v_1, \ldots, v_m \, \varphi$, where φ is in 3DNF. Checking validity of such formulas is known to be Σ_2^{P}-complete. For the reduction, we consider the following query $Q = Q_{\mathsf{VAL}} \wedge Q_s$, where Q_{VAL} is the $\exists\mathsf{FO}$ query from Theorem 6 that encodes the validity conditions, and Q_s is a $\forall\mathsf{FO}$ query defined as

$$Q_s := \forall x \, (\neg \mathsf{E}(x) \wedge \mathsf{L}(x)) \vee (\mathsf{E}(x) \wedge \neg \mathsf{L}(x)) \vee \mathsf{F}(x).$$

Moreover, we define a PDB $\mathcal{P}_\Phi$ such that

- for each variable u that appears in φ, $\mathcal{P}_\Phi$ contains the atom $\langle \mathsf{L}(u) : 0.5 \rangle$;
- for each existentially quantified variable u_j, $\mathcal{P}_\Phi$ has the atom $\langle \mathsf{E}(u_j) : 0.5 \rangle$;
- for every universally quantified variable v_j, $\mathcal{P}_\Phi$ contains the atom $\langle \mathsf{F}(v_j) : 1 \rangle$;
- every conjunction in φ is described with the help of the predicates R_1, …, R_4, each of which corresponds to one type of conjunctive clause. For example, if we have $\varphi_j = u_1 \wedge \neg u_2 \wedge \neg u_4$, we add the atom $\langle \mathsf{R}_3(u_4, u_2, u_1) : 1 \rangle$ to $\mathcal{P}_\Phi$, which enforces via Q_{VAL} that the clause that includes all atoms $\neg\mathsf{L}(u_4)$, $\neg\mathsf{L}(u_2)$, and $\mathsf{L}(u_1)$ can be true. Moreover, all the remaining R_i-atoms have probability 0.

In this construction, Q_{VAL} encodes the 3DNF, and Q_s helps us to distinguish between the existentially and universally quantified variables through the E- and F-atoms.

Claim. The quantified Boolean formula Φ is valid if and only if there exists a hypothesis $\mathcal{H}$ over $\mathcal{P}_\Phi$ such that $\mathrm{P}(\mathcal{H}) \geq (0.5)^{2n}$ and $\mathcal{H} \models Q$.

Suppose that Φ is valid. Then, there exists a valuation μ of $u_1, \ldots, u_n$, such that all valuations τ that extend this partial valuation (by assigning truth values to $v_1, \ldots, v_m$) satisfy φ. We define a hypothesis $\mathcal{H}$ depending on μ as follows. For all assignments $u_j \mapsto 1$ in μ, we add $\mathsf{L}(u_j)$ to $\mathcal{H}$; if, on the other hand, $u_j \mapsto 0$ in μ, we add $\neg\mathsf{L}(u_j)$ to $\mathcal{H}$. Moreover, to satisfy the query Q_s, for every $\mathsf{L}(u_j) \in \mathcal{H}$, we add $\neg\mathsf{E}(u_j)$ to $\mathcal{H}$, and analogously, for every $\neg\mathsf{L}(u_j) \in \mathcal{H}$, we add $\mathsf{E}(u_j)$ to $\mathcal{H}$. By this construction, there are clearly $2n$ atoms in $\mathcal{H}$, each of which has the probability 0.5 in $\mathcal{P}_\varphi$. Hence, it holds that $\mathrm{P}(\mathcal{H}) = (0.5)^{2n}$. Finally, it is sufficient to observe that all databases $\mathcal{D}$ that extend $\mathcal{H}$ must satisfy the query Q, as every such database is in one-to-one correspondence with a valuation τ that extends μ.

For the other direction, we assume that there exists a hypothesis $\mathcal{H}$ over $\mathcal{P}_\Phi$ such that $\mathrm{P}(\mathcal{H}) \geq (0.5)^{2n}$ and $\mathcal{H} \models Q$. This implies that $\mathcal{H}$ contains at most $2n$ atoms that have probability 0.5 in $\mathcal{P}_\varphi$ (and possibly some deterministic atoms). Furthermore, since $\mathcal{H} \models Q_s$, we know that $\mathcal{H}$ contains each E-atom either positively or negatively, and it also contains the complementary L-atom. Since these are already $2n$ atoms, $\mathcal{H}$ cannot contain any L-atoms for the universally quantified variables v_j. We can thus define a valuation μ for $u_1, \ldots, u_n$ simply by setting $u_j \mapsto 1$, if $\mathsf{L}(u_j) \in \mathcal{H}$, and $u_j \mapsto 0$, if $\neg\mathsf{L}(u_j) \in \mathcal{H}$. It is easy to see that the extensions τ of μ are in one-to-one correspondence with the databases that extend $\mathcal{H}$, and that φ evaluates to true for all of these assignments. □

With this result, we conclude the data complexity analysis for MPH. We first showed a data tractability result concerning unions of conjunctive queries analogous to MPD. As before, this is the only result with no matching lower bound. Unlike MPD, however, ∃FO queries are proven to be coNP-complete for MPH in data complexity. Besides, MPH remains NP-complete for ∀FO queries, but turns out to be Σ_2^{P}-complete for first-order queries in data complexity.

4 Probabilistic Knowledge Bases

We have discussed several limitations of PDBs in the introduction, and stated that the unrealistic assumptions employed in PDBs lead to even more undesired consequences once combined with another limitation of these systems, namely, the lack of *commonsense knowledge*, a natural component of human reasoning, which is not present in plain (probabilistic) databases. A common way of encoding commonsense knowledge is in the form of ontologies.

In this section, we enrich probabilistic databases with ontological knowledge. More precisely, we adopt the terminology from [7] and speak of ontology-mediated queries (OMQs), that is, database queries (typically, unions of conjunctive queries) coupled with an ontology. The task of evaluating such queries is then called *ontology-mediated query answering (OMQA)*. We first give a brief overview on the Datalog$^{\pm}$ family of languages, and introduce the paradigm of ontology-mediated query answering. Afterwards, we present results regarding ontology-mediated query evaluation over PDBs. We also discuss maximal posterior reasoning problems in the context of ontology-mediated queries.

4.1 Ontology-Mediated Query Answering in Datalog$^{\pm}$

We again consider a relational vocabulary σ, which is now extended with a (potentially infinite) set $\mathbf{N}$ of nulls. We first introduce the so-called *negative constraints (NCs)*. From a database perspective, NCs can be seen as a special case of denial constraints over databases [67]. Formally, a *negative constraint (NC)* is a first-order formula of the form $\forall \boldsymbol{x}\, \Phi(\boldsymbol{x}) \rightarrow \bot$, where $\Phi(\boldsymbol{x})$ is a conjunction of atoms, called the *body* of the NC, and $\bot$ is the truth constant *false*. Consider, for example, the NCs

$$\forall x\, \mathsf{Writer}(x) \wedge \mathsf{Novel}(x) \rightarrow \bot,$$
$$\forall x, y\, \mathsf{ParentOf}(x, y) \wedge \mathsf{ParentOf}(y, x) \rightarrow \bot.$$

The former states that *writers* and *novels* are disjoint entities, whereas the latter asserts that the ParentOf relation is antisymmetric.

To formulate more general ontological knowledge, tuple-generating dependencies are introduced. Intuitively, such dependencies describe constraints on databases in the form of generalized Datalog rules with existentially quantified conjunctions of atoms in rule heads. Formally, a *tuple-generating dependency (TGD)* is a first-order formula of the form $\forall \boldsymbol{x}\, \Phi(\boldsymbol{x}) \rightarrow \exists \boldsymbol{y}\, \Psi(\boldsymbol{x}, \boldsymbol{y})$, where

$\Phi(\boldsymbol{x})$ is a conjunction of atoms, called the *body* of the TGD, and $\Psi(\boldsymbol{x}, \boldsymbol{y})$ is a conjunction of atoms, called the *head* of the TGD. Consider the TGDs

$$\forall x, y\, \mathsf{AuthorOf}(x, y) \wedge \mathsf{Novel}(y) \rightarrow \mathsf{Writer}(x), \tag{1}$$

$$\forall y\, \mathsf{Novel}(y) \rightarrow \exists x\, \mathsf{AuthorOf}(x, y) \wedge \mathsf{Writer}(x). \tag{2}$$

The first one states that anyone who authors a novel is a writer. The second one asserts that all novels are authored by a writer. Note that TGDs can express the well-known inclusion dependencies and join dependencies from database theory. A *Datalog*$^{\pm}$ *program* (or *ontology*) Σ is a finite set of negative constraints and tuple generating dependencies. A Datalog$^{\pm}$ program is *positive* if it consists of only TGDs, i.e., does not contain any NCs.

In essence, Datalog$^{\pm}$ languages are only syntactic fragments of first-order logic, which also employ the *standard name assumption*, as in databases. Thus, a first-order interpretation $\mathcal{I}$ is a model of an ontology Σ in the classical sense, i.e., if $\mathcal{I} \models \alpha$ for all $\alpha \in \Sigma$. Given a database $\mathcal{D}$ defined over known constants and an ontology Σ, we write $mods(\Sigma, \mathcal{D})$ to represent the set of models of Σ that extend $\mathcal{D}$, which is formally defined as $\{\mathcal{I} \mid \mathcal{I} \models \mathcal{D}, \mathcal{I} \models \Sigma\}$. A database $\mathcal{D}$ is *consistent* w.r.t. Σ if $mods(\Sigma, \mathcal{D})$ is non-empty.

The entailment problem in Datalog$^{\pm}$ ontologies is undecidable [5], which motivated syntactic restrictions on Datalog$^{\pm}$ ontologies; there are a plethora of classes of TGDs [11, 4, 45, 10, 28]; here, we only focus on some of these classes. Our choice is primarily for ease of presentation, and most of the results can be extended to other classes in a straight-forward manner; see e.g. [13], for a more detailed analysis.

The (syntactic) restrictions on TGDs that we recall are guardedness [10], stickiness [12], and acyclicity, along with their "weak" counterparts, weak guardedness [10], weak stickiness [12], and weak acyclicity [28], respectively. A TGD is *guarded*, if there exists a body atom that contains (or "guards") all body variables. The class of guarded TGDs, denoted G, is defined as the family of all possible sets of guarded TGDs. A key subclass of guarded TGDs are the *linear* TGDs with just one body atom, which is automatically the guard. The class of linear TGDs is denoted by L. *Weakly guarded* TGDs extend guarded TGDs by requiring only the body variables that are considered "harmful" to appear in the guard (see [10] for full details). The associated class of TGDs is denoted WG. It is easy to verify that $\mathsf{L} \subset \mathsf{G} \subset \mathsf{WG}$.

Stickiness is inherently different from guardedness, and its central property can be described as follows: variables that appear more than once in a body (i.e., join variables) must always be propagated (or "stuck") to the inferred atoms. A TGD that enjoys this property is called *sticky*, and the class of sticky TGDs is denoted by S. Weak stickiness generalizes stickiness by considering only "harmful" variables, and defines the class WS of *weakly sticky* TGDs. Observe that $\mathsf{S} \subset \mathsf{WS}$.

A set of TGDs is *acyclic* and belongs to the class A if its predicate graph is acyclic. Equivalently, an acyclic set of TGDs can be seen as a non-recursive set of TGDs. A set of TGDs is *weakly acyclic*, if its dependency graph enjoys a

Table 6. A database that consists of the unary relations Writer, Novel and the binary relation AuthorOf.

Writer	AuthorOf		Novel
balzac	hamsun	hunger	goriot
dostoyevski	dostoyevski	gambler	hunger
kafka	kafka	trial	trial

certain acyclicity condition, which guarantees the existence of a finite canonical model; the associated class is denoted WA. Clearly, $\mathsf{A} \subset \mathsf{WA}$. Interestingly, it also holds that $\mathsf{WA} \subset \mathsf{WS}$ [12].

Another fragment of TGDs are *full* TGDs, i.e., TGDs without existentially quantified variables. The corresponding class is denoted by F. Restricting full TGDs to satisfy linearity, guardedness, stickiness, or acyclicity yields the classes LF, GF, SF, and AF, respectively. It is known that $\mathsf{F} \subset \mathsf{WA}$ [28] and $\mathsf{F} \subset \mathsf{WG}$ [10].

We usually omit the universal quantifiers in TGDs and NCs, and for clarity we consider single-atom-head TGDs; however, our results can be easily extended to TGDs with conjunctions of atoms in the head (except under the bounded-arity assumption). Following the common convention, we will assume that NCs are part of all Datalog$^{\pm}$ languages.

Ontology-mediated query answering is a popular paradigm for querying incomplete data sources in a more adequate manner [7]. Formally, an *ontology-mediated query (OMQ)* is a pair $(Q, \mathcal{T})$, where Q is a Boolean query, and $\mathcal{T}$ is an ontology. Given a database $\mathcal{D}$ and an OMQ $(Q, \mathcal{T})$, we say that $\mathcal{D}$ entails the OMQ $(Q, \mathcal{T})$, denoted $\mathcal{D} \models (Q, \mathcal{T})$, if for all models $\mathcal{I} \models (\mathcal{T}, \mathcal{D})$ it holds that $\mathcal{I} \models Q$. Then, *ontology-mediated query answering (OMQA)* is the task of deciding whether $\mathcal{D} \models (Q, \mathcal{T})$ for a given database $\mathcal{D}$ and an OMQ $(Q, \mathcal{T})$. Note that we use the term query answering in a rather loose sense to refer to the Boolean query evaluation problem.

Example 9. Let us consider the database $\mathcal{D}_a$ given in Table 6. Observe that the simple queries

$$Q_1 \coloneqq \mathsf{Writer}(\mathsf{hamsun}) \quad \textit{and} \quad Q_2 \coloneqq \exists x\, \mathsf{Writer}(x) \wedge \mathsf{AuthorOf}(x, \mathsf{goriot})$$

are not satisfied by the database $\mathcal{D}_a$ although they should evaluate to true from an intuitive perspective. On the other side, under the Datalog$^{\pm}$ program Σ_a that consists of the TGDs

$$\forall x, y\, \mathsf{AuthorOf}(x, y) \wedge \mathsf{Novel}(y) \rightarrow \mathsf{Writer}(x),$$
$$\forall y\, \mathsf{Novel}(y) \rightarrow \exists x\, \mathsf{AuthorOf}(x, y) \wedge \mathsf{Writer}(x),$$

both of these queries are satisfied: $\mathcal{D}_a \models (Q_1, \Sigma_a)$ holds due to the first rule, and $\mathcal{D}_a \models (Q_2, \Sigma_a)$ holds due to the second rule. The incomplete database is queried through the logical rules that encode commonsense knowledge, which in turn results in more complete answers. ■

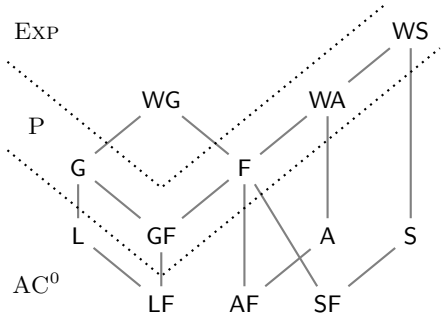


Fig. 4. Inclusion relationships and data complexity of ontology-mediated query answering for Datalog$^{\pm}$ languages.

A key paradigm in ontology-mediated query answering is the *first-order rewritability* of queries. Intuitively, FO-rewritability ensures that we can rewrite an OMQ into a (possibly large) UCQ, and this transformation is *homomorphism-preserving over all finite structures* [62]. More formally, let $\mathcal{T}$ be an ontology and Q a Boolean query. Then, the OMQ $(Q, \mathcal{T})$ is *FO-rewritable* if there exists a Boolean UCQ $Q_{\mathcal{T}}$ such that, for all databases $\mathcal{D}$ that are consistent w.r.t. $\mathcal{T}$, we have $\mathcal{D} \models (Q, \mathcal{T})$ if and only if $\mathcal{D} \models Q_{\mathcal{T}}$. In this case, $Q_{\mathcal{T}}$ is called an *FO-rewriting* of $(Q, \mathcal{T})$. A language $\mathcal{L}$ is *FO-rewritable* if it admits an FO-rewriting for any UCQ and theory in $\mathcal{L}$.

FO-rewritability implies a data-independent reduction from OMQA to query evaluation in relational databases. In practical terms, this means that the query can be rewritten into an SQL query to be evaluated in relational database management systems. In theoretical terms, this puts OMQA in AC^0 in data complexity for all FO-rewritable languages. The data complexity of ontology-mediated query answering for basic Datalog$^{\pm}$ languages is summarized in Figure 4.

4.2 Ontology-Mediated Queries for Probabilistic Databases

We give an overview of the problem of evaluating ontology-mediated queries for PDBs. The idea is to allow Datalog$^{\pm}$ programs on top of tuple-independent PDBs and query the probability of a given OMQ.

Definition 11 (semantics). The *probability of an OMQ* (Q, Σ) *relative to a probability distribution* P is

$$\mathrm{P}(Q, \Sigma) = \sum_{\mathcal{D} \models (Q, \Sigma)} \mathrm{P}(\mathcal{D}),$$

where $\mathcal{D}$ ranges over all databases over σ.

The major difference compared to PDBs is that this semantics defers the decision of whether a world satisfies a query to an entailment test, which also includes a logical theory.

Table 7. A probabilistic database $\mathcal{P}_a$, which consists of the unary relations Writer, Novel and the binary relation AuthorOf.

Writer	P
balzac	0.8
dostoyevski	0.6
kafka	0.9

AuthorOf		P
hamsun	hunger	0.9
dostoyevski	gambler	0.6
kafka	trial	0.8

Novel	P
goriot	0.7
hunger	0.4
trial	0.5

Note that the Datalog$^{\pm}$ program can be inconsistent with some of the worlds, which makes standard reasoning very problematic, as *anything* can be entailed from an inconsistent theory (*"ex falso quodlibet"*). The common way of tackling this problem in probabilistic knowledge bases is to restrict probabilistic query evaluation to only consider consistent worlds by setting the probabilities of inconsistent worlds to 0 and *renormalizing* the probability distribution over the set of worlds accordingly. More formally, the query probabilities can be normalized by defining

$$\mathrm{P}_n(Q, \Sigma) \coloneqq (\mathrm{P}(Q, \Sigma) - \gamma)/(1 - \gamma),$$

where γ is the probability of the inconsistent worlds given as

$$\gamma \coloneqq \sum_{mods(\Sigma, \mathcal{D}) = \emptyset} \mathrm{P}(\mathcal{D}).$$

The normalization factor γ can thus be computed once and then reused as a post-processing step. Hence, for simplicity, we assume that all the worlds $\mathcal{D}$ induced by the PDB are consistent with the program.

Let us now illustrate the effect of ontological rules in PDBs. Recall Example 9, where we illustrated the effect of ontological rules in querying databases. Queries that intuitively follow from the knowledge encoded in the database were not satisfied by the given database (from Table 6). Still, it was possible to alleviate this problem using ontological rules. We now adopt this example to PDBs and observe a similar effect.

Example 10. Let us consider the PDB $\mathcal{P}_a$ given in Table 7. The queries

$$Q_1 \coloneqq \mathsf{Writer}(\mathsf{hamsun}) \quad \textit{and} \quad Q_2 \coloneqq \exists x\, \mathsf{Writer}(x) \wedge \mathsf{AuthorOf}(x, \mathsf{goriot})$$

from Example 9 evaluate to the probability 0 on $\mathcal{P}_a$. On the other side, under the Datalog$^{\pm}$ program Σ_a that consists of the rules

$$\forall x, y\, \mathsf{AuthorOf}(x, y) \wedge \mathsf{Novel}(y) \rightarrow \mathsf{Writer}(x),$$
$$\forall y\, \mathsf{Novel}(y) \rightarrow \exists x\, \mathsf{AuthorOf}(x, y) \wedge \mathsf{Writer}(x),$$

there are worlds where both of these queries are satisfied. One such world is $\mathcal{D}_a$ given in Table 6, i.e., recall that $\mathcal{D}_a \models (Q_1, \Sigma_a)$ and $\mathcal{D}_a \models (Q_2, \Sigma_a)$. More precisely, we obtain that $\mathrm{P}(Q_1) = 0.63$, since any world that contains both $\mathsf{AuthorOf}(\mathsf{hamsun}, \mathsf{hunger})$ and $\mathsf{Novel}(\mathsf{hunger})$ entails $\mathsf{Writer}(\mathsf{hamsun})$, and no other world does. Similarly, $\mathrm{P}(Q_2) = 0.7$, since Q_2 is entailed from all and only those worlds where $\mathsf{Novel}(\mathsf{goriot})$ holds. ■

Probabilistic query evaluation, as a decision problem, is defined as before with the only difference that it is now parametrized with OMQs.

Definition 12 (probabilistic query evaluation). Given a PDB $\mathcal{P}$, an OMQ (Q, Σ), and a value $p \in [0, 1)$, *probabilistic query evaluation*, denoted PQE, is to decide whether $\mathrm{P}_{\mathcal{P}}(Q, \Sigma) > p$ holds. PQE is parametrized with the language of the ontology and the query; we write $\mathsf{PQE}(\mathsf{Q}, \mathcal{L})$ to define PQE on the class Q of queries and on the class of ontologies restricted to the language $\mathcal{L}$.

We will deliberately use the terms probabilistic query evaluation and probabilistic OMQ evaluation interchangeably if there is no danger of ambiguity. We now provide a host of complexity results for probabilistic OMQ evaluation relative to different languages.

We start our complexity analysis with a rather simple result that is of a generic nature. Intuitively, given the complexity of OMQA in a Datalog$^{\pm}$ language, we obtain an immediate upper and lower bound for the complexity of probabilistic OMQ evaluation in that language.

Theorem 9. *Let $\mathfrak{C}$ denote the data complexity of ontology-mediated query answering for a Datalog$^{\pm}$ language $\mathcal{L}$. Then, $\mathsf{PQE}(\mathsf{UCQ}, \mathcal{L})$ is $\mathfrak{C}$-hard and in $\mathrm{PP}^{\mathfrak{C}}$ for PDBs in data complexity.*

Proof. Let (Q, Σ) be an OMQ, $\mathcal{P}$ be a PDB, and $p \in [0, 1)$ be a threshold value. Consider a nondeterministic Turing machine with a $\mathfrak{C}$ oracle. Each branch corresponds to a world $\mathcal{D}$ and is marked as either accepting or rejecting depending on the outcome of the logical entailment check $\mathcal{D} \models (Q, \Sigma)$. This logical entailment test is in $\mathfrak{C}$ for the language $\mathcal{L}$, by our assumption. Thus, it can be performed using the oracle. Then, by this construction, the nondeterministic Turing machine answers yes if and only if $\mathrm{P}_{\mathcal{P}}(Q, \Sigma) > p$, which proves membership to $\mathrm{PP}^{\mathfrak{C}}$ in the respective complexity.

To show $\mathfrak{C}$-hardness, we reduce from ontology-mediated query answering, that is, given a database $\mathcal{D}$ and an OMQ (Q, Σ), where Q is a UCQ, and Σ is a program over $\mathcal{L}$, decide whether $\mathcal{D} \models (Q, \Sigma)$. We define a PDB $\mathcal{P}$ that contains all the atoms from the database $\mathcal{D}$ with probability 1. Then, it is easy to see that $\mathcal{D} \models (Q, \Sigma)$ if and only if $\mathrm{P}_{\mathcal{P}}(Q) \geq 1$. □

We briefly analyze the consequences of Theorem 9. Observe that, for all deterministic complexity classes $\mathfrak{C}$ that contain PP, it holds that $\mathrm{PP}^{\mathfrak{C}} = \mathfrak{C}$, and thus Theorem 9 directly implies tight complexity bounds. For instance, the data complexity of probabilistic OMQ evaluation for WG is Exp-complete as a simple consequence of Theorem 9.

Beyond this generic result, one is interested in lifting the data complexity dichotomy for unions of conjunctive queries to OMQs. This connection is immediate, as shown in the following.

Lemma 1. *Let (Q, Σ) be an OMQ, where Q is a UCQ, and Σ is a program, and Q_{Σ} be an FO-rewriting of (Q, Σ). Then, for any PDB $\mathcal{P}$, it holds that $\mathrm{P}_{\mathcal{P}}(Q, \Sigma) = \mathrm{P}_{\mathcal{P}}(Q_{\Sigma})$.*

Proof. For any PDB $\mathcal{P}$, it holds that

$$\mathrm{P}_{\mathcal{P}}(Q,\Sigma) \overset{(1)}{=} \sum_{\mathcal{D}\models(Q,\Sigma)} \mathrm{P}_{\mathcal{P}}(\mathcal{D}) \overset{(2)}{=} \sum_{\mathcal{D}\models Q_\Sigma} \mathrm{P}_{\mathcal{P}}(\mathcal{D}) \overset{(3)}{=} \mathrm{P}_{\mathcal{P}}(Q_\Sigma),$$

where (1) follows from Definition 11; (2) follows from Q_Σ being the FO-rewriting of Q w.r.t. Σ; and (3) is the definition of the semantics of Q_Σ in PDBs. □

With the help of Lemma 1, it becomes possible to lift the data complexity dichotomy in probabilistic databases to all Datalog$^\pm$ languages that are FO-rewritable.

Theorem 10 (dichotomy). *For all FO-rewritable Datalog$^\pm$ languages $\mathcal{L}$, the following holds.* $\mathsf{PQE}(\mathsf{UCQ},\mathcal{L})$ *is either in* P *or it is* PP*-complete for PDBs in data complexity under polynomial-time Turing reductions.*

Proof. Let (Q,Σ) be an OMQ, where Q is a UCQ, and Σ is a Datalog$^\pm$ program over an FO-rewritable language, and Q_Σ be an FO-rewriting of (Q,Σ). By Lemma 1, any polynomial-time algorithm that can evaluate Q_Σ over PDBs also yields the probability of the OMQ (Q,Σ) relative to an PDB, and vice versa. This implies that the OMQ (Q,Σ) is safe if Q_Σ is safe.

Dually, by the same result, the probabilities of *all* rewritings of Q coincide, and hence the same algorithm can be used for all of them. Thus, if (Q,Σ) is unsafe, then Q_Σ must also be unsafe for PDBs. By the dichotomy of [21] and Lemma 1, this implies that evaluating the probability of both the UCQ Q_Σ and the OMQ (Q,Σ) must be PP-hard under Turing reductions. □

Obviously, ontological rules introduce dependencies. Therefore, a safe query can become unsafe for OMQs. However, the opposite effect is also possible, i.e., an unsafe query may become safe under ontological rules. We illustrate both of these effects on a synthetic example.

Example 11. Consider the conjunctive query $\exists x,y\,\mathsf{C}(x) \wedge \mathsf{D}(x,y)$, which is safe for PDBs. It becomes unsafe under the TGD $\mathsf{R}(x,y), \mathsf{T}(y) \rightarrow \mathsf{D}(x,y)$, since then it rewrites to the query

$$(\exists x,y\,\mathsf{C}(x) \wedge \mathsf{D}(x,y)) \vee (\exists x,y\,\mathsf{C}(x) \wedge \mathsf{R}(x,y) \wedge \mathsf{T}(y)),$$

which is unsafe. Conversely, the conjunctive query $\exists x,y\,\mathsf{C}(x) \wedge \mathsf{R}(x,y) \wedge \mathsf{D}(y)$ is not safe for PDBs, but becomes safe under the TGD $\mathsf{R}(x,y) \rightarrow \mathsf{D}(y)$, as it rewrites to $\exists x,y\,\mathsf{C}(x) \wedge \mathsf{R}(x,y)$. Note that these are very simple TGDs, which are full, acyclic, guarded, and sticky. ■

Recall that the PP-hardness of probabilistic UCQ evaluation in data complexity holds under polynomial-time Turing reductions. This transfers to probabilistic OMQ evaluation relative to FO-rewritable languages. On the other hand, for guarded full programs, it is possible to show that PP-hardness holds even under standard many-one reductions.

Theorem 11. $\mathsf{PQE}(\mathsf{UCQ}, \mathsf{GF})$ *is* PP*-hard for PDBs in data complexity.*

Proof. We reduce the following problem [74]: decide the validity of the formula $\Phi = \mathsf{C}^c\, x_1, \ldots, x_n\ \varphi$, where $\varphi = \varphi_1 \wedge \cdots \wedge \varphi_k$ is a propositional formula in CNF over the variables $x_1, \ldots, x_n$. This amounts to checking whether there are at least c assignments to $x_1, \ldots, x_n$ that satisfy φ. We assume without loss of generality that φ contains all clauses of the form $x_j \vee \neg x_j$, $1 \leq j \leq n$; clearly, this does not affect the existence or number of satisfying assignments for φ. For the reduction, we use a PDB $\mathcal{P}_\Phi$ and a program Σ_Φ. We first define the PDB $\mathcal{P}_\Phi$ as follows.

- For each variable x_j, $1 \leq j \leq n$, $\mathcal{P}_\Phi$ contains the probabilistic atoms $\langle \mathsf{L}(x_j, 0) : 0.5 \rangle$ and $\langle \mathsf{L}(x_j, 1) : 0.5 \rangle$, where we view x_j as a constant. These atoms represent the assignments that map x_j to *false* and *true*, respectively.
- For each propositional literal $(\neg)x_\ell$ occurring in a clause φ_j, $1 \leq j \leq k$, $\mathcal{P}_\Phi$ contains the atom $\mathsf{D}(x_\ell, j, i)$ with probability 1, where $i = 1$, if the literal is positive, and $i = 0$, if the literal is negative.
- $\mathcal{P}_\Phi$ contains all the atoms $\mathsf{T}(0)$, $\mathsf{S}(0,1), \mathsf{S}(1,2), \ldots, \mathsf{S}(k-1, k)$, $\mathsf{K}(k)$, each with probability 1.

We now describe the program Σ_Φ. To detect when a clause is satisfied, we use the additional unary predicate E and the TGD

$$\mathsf{L}(x, i), \mathsf{D}(y, j, i) \rightarrow \mathsf{E}(j),$$

which is a universally quantified formula over the variables x, y, i, and j. We still need to ensure that in each world, exactly one of $\mathsf{L}(x, 0)$ and $\mathsf{L}(x, 1)$ holds. The clauses $x_j \vee \neg x_j$ take care of the lower bound; for the variables $x_1, \ldots, x_n$, we use the TGDs

$$\mathsf{L}(x, 0), \mathsf{L}(x, 1) \rightarrow \mathsf{B} \qquad \text{and} \qquad \mathsf{B}, \mathsf{D}(y, j, i) \rightarrow \mathsf{E}(j).$$

These TGDs ensure that any inconsistent assignment for $x_1, \ldots, x_n$, i.e., one where some x_j is both *true* and *false*, is automatically marked as satisfying the formula, even if the clause $x_j \vee \neg x_j$ is actually not satisfied. Since there are exactly $4^n - 3^n$ such assignments (where both $\mathsf{L}(x_j, 0)$ and $\mathsf{L}(x_j, 1)$ hold for at least one x_j), we can add this number to the probability threshold that we will use in the end. Note that the probability of each individual assignment is 0.25^n, since there are $2n$ relevant L-atoms (the other atoms are fixed to 0 or 1 and do not contribute here).

It remains to detect whether *all* clauses of φ are satisfied by a consistent assignment, which we do by the means of the TGDs

$$\mathsf{T}(i), \mathsf{S}(i, j), \mathsf{E}(j) \rightarrow \mathsf{T}(j) \qquad \text{and} \qquad \mathsf{T}(i), \mathsf{K}(i) \rightarrow \mathsf{Z}(i).$$

Lastly, we define the simple UCQ $Q \coloneqq \exists i\ \mathsf{Z}(i)$. Then, we prove the following claim.

Claim. $\mathrm{P}_{\mathcal{P}_\Phi}(Q, \Sigma_\Phi) \geq 0.25^n(4^n - 3^n + c)$ holds if and only if Φ is valid.

Suppose that Φ is valid, i.e., there are at least c different assignments to $x_1, \ldots, x_n$ that satisfy φ. Then, for each such assignment τ, we can define a world $\mathcal{D}_\tau$ such that it contains all atoms from $\mathcal{P}_\Phi$ that occur with probability 1. Moreover, $\mathcal{D}_\tau$ contains an atom $\mathsf{L}(x_j, 1)$, if x_j is mapped to true in μ, and an atom $\mathsf{L}(x_j, 0)$, if x_j is mapped to false in μ. It is easy to see that each such database $\mathcal{D}_\tau$ is induced by the PDB $\mathcal{P}_\Phi$ and that $\mathcal{D}_\tau \models Q$ by our constructions. In particular, this implies that $\mathcal{D}_\tau \models Q_\Phi$ for c worlds. Recall also that $(4^n - 3^n)$ worlds, capturing the inconsistent valuations, satisfy the query. As every world has the probability $(0.5)^{2n}$, we conclude that $\mathrm{P}_{\mathcal{P}_\Phi}(Q_\Phi) \geq 0.5^{2n}(4^n - 3^n + c)$.

Conversely, if the query probability exceeds the threshold value, then some worlds in $\mathcal{P}_\Phi$ with non-zero probability entail (Q, Σ_Φ), i.e., all clauses of φ are satisfied. Each of the non-zero worlds in PDBs represents a unique combination of atoms of the form $\mathsf{L}(x, 0)$ and $\mathsf{L}(x, 1)$. The worlds where for at least one variable x_j, $1 \leq j \leq n$, neither $\mathsf{L}(x_j, 0)$ nor $\mathsf{L}(x_j, 1)$ holds do not satisfy φ, and hence do not entail (Q, Σ_Φ) and are not counted. Excluding $(4^n - 3^n)$ worlds capturing the inconsistent valuations, all other worlds represent the actual assignments for $x_1, \ldots, x_n$, and hence we know that at least c of those satisfy φ. Thus, we conclude that Φ is valid.

Observe that all TGDs used in the reduction are *full* and *guarded*. Moreover, only the PDB and the probability threshold depend on the input formula (which is allowed in data complexity). Hence, the reduction shows PP-hardness of $\mathsf{PQE}(\mathsf{GF}, \mathsf{UCQ})$ for PDBs in data complexity. □

GF is one of the least expressive Datalog$^\pm$ languages with polynomial time data complexity for OMQA. Thus, this result already implies PP-hardness for the classes G, F, WS, and WA. This completes all results regarding the data complexity. It remains open whether the data complexity dichotomy can be extended to Datalog-rewritable languages. Clearly, a data complexity dichotomy in these languages would be closely related to a similar result in Datalog.

4.3 Most Probable Explanations for Ontology-Mediated Queries

Motivated by maximal posterior computations in PGMs, we studied the most probable database and most probable hypothesis problems in PDBs. We now extend these results towards ontology-mediated queries. In a nutshell, we restrict ourselves to unions of conjunctive queries (instead of first-order queries), but in exchange consider additional knowledge encoded through an ontology.

Importantly, in MPD (resp., MPH), we are interested in finding the database (resp., the hypothesis) that maximizes the query probability. In the presence of ontological rules, we need to ensure that the chosen database is at the same time consistent with the ontology. More precisely, for ontology-mediated queries, models must be consistent with the ontology; thus, the definitions of MPD and MPH are adapted accordingly.

The Most Probable Database Problem for Ontological Queries. In the most probable database problem for ontological queries, we consider only the

Table 8. The probabilistic database $\mathcal{P}_v$.

Vegetarian		FriendOf		Eats		Meat	
alice	0.7	alice bob	0.7	bob spinach	0.7	shrimp	0.7
bob	0.9	alice chris	0.8	chris mussels	0.8	mussels	0.9
chris	0.6	bob chris	0.1	alice broccoli	0.2	seahorse	0.3

consistent worlds induced by the PDB, and thus maximize only over consistent worlds.

Definition 13. Let $\mathcal{P}$ be a probabilistic database and Q a query. The *most probable database* for an OMQ (Q, Σ) over a PDB $\mathcal{P}$ is given by

$$\underset{\mathcal{D} \models (Q,\Sigma), mods(\mathcal{D},\Sigma) \neq \emptyset}{\arg\max} \mathrm{P}(\mathcal{D}),$$

where $\mathcal{D}$ ranges over all worlds induced by $\mathcal{P}$.

To illustrate the semantics, we now revisit the Example 4 and the PDB $\mathcal{P}_v$, which is depicted in Table 8.

Example 12. Recall the following query

$$Q_{\mathsf{veg}} \coloneqq \forall x, y \; \neg \mathsf{Vegetarian}(x) \vee \neg \mathsf{Eats}(x, y) \vee \neg \mathsf{Meat}(y).$$

The most probable database for Q_{veg} contains all atoms from $\mathcal{P}_v$ that have a probability above 0.5, except for $\mathsf{Vegetarian}(\mathsf{chris})$. We can impose the same constraint through the negative constraint

$$\forall x, y \; \mathsf{Vegetarian}(x) \wedge \mathsf{Eats}(x, y) \wedge \mathsf{Meat}(y) \rightarrow \bot,$$

and then the most probable database for the query $\top$ will be the same as before. Obviously, we can additionally impose constraints in the form of TGDs, such as

$$\forall x \; \mathsf{Vegetarian}(x) \rightarrow \exists y \; \mathsf{FriendOf}(x, y) \wedge \mathsf{Vegetarian}(y),$$

which states that vegetarians have friends who are themselves vegetarians. ■

The corresponding decision problem is defined as before with the only difference that now we consider ontology-mediated queries.

Definition 14 (MPD). Let (Q, Σ) be an OMQ, $\mathcal{P}$ a probabilistic database, and $p \in (0, 1]$ a threshold. MPD is the problem of deciding whether there exists a database $\mathcal{D}$ that entails (Q, Σ) with $\mathrm{P}(\mathcal{D}) > p$. MPD is parametrized with the language of the ontology and the query; we write $\mathsf{MPD}(\mathsf{Q}, \mathcal{L})$ to define MPD on the class Q of queries and on the class of ontologies restricted to the languages $\mathcal{L}$.

We start our complexity analysis with some simple observations. Note first that consistency of a database $\mathcal{D}$ with respect to a program Σ can be written as $\mathcal{D} \not\models (\bot, \Sigma)$, or equivalently, $\mathcal{D} \not\models (Q_\bot, \Sigma^+)$, where $Q_\bot$ is the UCQ obtained from the disjunction of the bodies of all NCs in Σ, and Σ^+ is the corresponding positive program (that contains all TGDs of Σ). This transformation allows us to rewrite the NCs into the query.

A naïve approach to solve MPD is to first guess a database $\mathcal{D}$, and then check that it entails the given OMQ does *not* entail the query $(Q_\bot, \Sigma^+)$ (i.e., it is consistent with the program), and exceeds the probability threshold. Since the probability can be computed in polynomial time, the problem can be decided by a nondeterministic Turing machine using an oracle to check OMQA. Obviously, MPD is at least as hard as OMQA in the underlying ontology languages. These observations result in the following theorem.

Theorem 12. *Let $\mathfrak{C}$ denote the data complexity of ontology-mediated query answering for a Datalog$^\pm$ language $\mathcal{L}$. $\mathsf{MPD}(\mathsf{UCQ}, \mathcal{L})$ is $\mathfrak{C}$-hard and in $\mathrm{NP}^{\mathfrak{C}}$ under the same complexity assumptions.*

By a reduction from 3-colorability, it is possible to show that MPD is NP-hard already in data complexity for OMQs, even if we only use NCs, i.e., the query and the positive program Σ^+ are empty. This strengthens the previous result about $\forall\mathsf{FO}$ queries, since NCs can be expressed by universal queries, but are not allowed to use negated atoms.

Theorem 13. $\mathsf{MPD}(\mathsf{UCQ}, \mathsf{NC})$ *is* NP*-complete in data complexity (which holds even for instance queries).*

Proof. The upper bound is easy to obtain; thus, we only show the lower bound. We provide a reduction from the well-known 3-colorability problem: given an undirected graph $G = (V, E)$, decide whether the nodes of G are 3-colorable. We first define the PDB $\mathcal{P}_G$ as follows. For all edges $(u, v) \in E$, we add the atom $\mathsf{E}(u, v)$ with probability 1, and for all nodes $u \in V$, we add the atoms $\mathsf{V}(u, 1)$, $\mathsf{V}(u, 2)$, $\mathsf{V}(u, 3)$, each with probability 0.7. In this encoding, the atoms $\mathsf{V}(u, 1)$, $\mathsf{V}(u, 2)$, $\mathsf{V}(u, 3)$ correspond to different colorings of the same node u.

We next define the conditions for 3-colorability through a set Σ containing only negative constraints (that do not depend on G). We need to ensure that each node is assigned *at most* one color, which is achieved by means of the negative constraints:

$$\mathsf{V}(x, 1), \mathsf{V}(x, 2) \rightarrow \bot, \quad \mathsf{V}(x, 1), \mathsf{V}(x, 3) \rightarrow \bot, \quad \mathsf{V}(x, 2), \mathsf{V}(x, 3) \rightarrow \bot.$$

Similarly, we need to enforce that the neighboring nodes are not assigned the same color, which we ensure with the negative constraint:

$$\mathsf{E}(x, y), \mathsf{V}(x, c), \mathsf{V}(y, c) \rightarrow \bot.$$

Finally, we define the query $Q \coloneqq \top$ and prove the following claim.

Claim. G is 3-colorable if and only if there exists a database $\mathcal{D}$ such that $\mathcal{D} \not\models (\bot, \Sigma)$ and $\mathrm{P}(\mathcal{D}) \geq (0.7 \cdot 0.3 \cdot 0.3)^{|V|}$.

Suppose that there exists a database $\mathcal{D}$ with a probability of at least $(0.7 \cdot 0.3 \cdot 0.3)^{|V|}$ that satisfies all NCs in Σ. Then, for every node $u \in V$, $\mathcal{D}$ must contain exactly one tuple $\mathtt{V}(u, c)$ for some color $c \in \{1, 2, 3\}$. Recall that *at most* was ensured by the first three NCs; hence, in order to achieve the given threshold, *at least* one of these tuples must be present. This yields a unique coloring for the nodes. Furthermore, since $\mathcal{D}$ must contain all tuples corresponding to the edges of G, and $\mathcal{D}$ satisfies the last NC, we conclude that G is 3-colorable.

Suppose that G is 3-colorable. Then, for a valid coloring, we define a DB $\mathcal{D}$ that contains all tuples that correspond to the edges, and add all tuples $\mathtt{V}(u, c)$ where c is the color of u. It is easy to see that $\mathcal{D}$ is consistent with Σ and $\mathrm{P}(\mathcal{D}) = (0.7 \cdot 0.3 \cdot 0.3)^{|V|}$, which concludes the proof. □

The Most Probable Hypothesis Problem for Ontological Queries. As before, we have to update the definition of most probable hypothesis to take into account only consistent worlds.

Definition 15. The *most probable hypothesis* for an OMQ (Q, Σ) over a PDB $\mathcal{P}$ is

$$\operatorname*{arg\,max}_{\mathcal{H} \models (Q, \Sigma)} \sum_{\substack{\mathcal{D} \supseteq \mathcal{H} \\ mods(\mathcal{D}, \Sigma) \neq \emptyset}} \mathrm{P}(\mathcal{D}),$$

where $\mathcal{H}$ is a set of (non-probabilistic) atoms t occurring in $\mathcal{P}$.

The corresponding decision problem is defined as before, but parametrized with ontology-mediated queries.

Definition 16 (MPH). Let (Q, Σ) be an OMQ, $\mathcal{P}$ a PDB, and $p \in (0, 1]$ a threshold. MPH is the problem of deciding whether there exists a hypothesis $\mathcal{H}$ that satisfies (Q, Σ) with $\mathrm{P}(\mathcal{H}) > p$. MPH is parametrized with the language of the ontology and the query; we write $\mathsf{MPH}(\mathsf{Q}, \mathcal{L})$ to define MPH on the class Q of queries and on the class of ontologies restricted to the languages $\mathcal{L}$.

Importantly, computing the probability of a hypothesis for an OMQ is not as easy as for classical database queries, since now there are also inconsistent worlds that shall be ruled out. As a consequence, computing the probability of a hypothesis becomes PP-hard, which makes a significant difference in the complexity analysis.

To solve MPH for OMQs, one can guess a hypothesis, and then check whether it entails the query, and whether the probability mass of its consistent extensions exceeds the given threshold. The latter part can be done by a PP Turing machine with an oracle for OMQA. The oracle can be used also for the initial entailment check. Clearly, $\mathsf{MPH}(\mathsf{UCQ}, \mathcal{L})$ is at least as hard as OMQA in $\mathcal{L}$. The next theorem follows from these observations.

Theorem 14. *Let $\mathfrak{C}$ denote the data complexity of ontology-mediated query answering for a Datalog$^{\pm}$ language $\mathcal{L}$. $\mathsf{MPD}(\mathsf{UCQ}, \mathcal{L})$ is $\mathfrak{C}$-hard and in $\mathrm{NP}^{\mathrm{PP}^{\mathfrak{C}}}$ under the same complexity assumptions.*

For any $\mathfrak{C} \subseteq \mathrm{PH}$, this result yields $\mathrm{NP}^{\mathrm{PP}^{\mathrm{PH}}} = \mathrm{NP}^{\mathrm{P}^{\mathrm{PP}}} = \mathrm{NP}^{\mathrm{PP}}$ as an upper bound, due to a result from [69]. For FO-rewritable Datalog$^{\pm}$ languages, we immediately obtain an $\mathrm{NP}^{\mathrm{PP}}$ upper bound for MPH as a consequence of Theorem 14. The obvious question is whether this is also a matching lower bound for FO-rewritable languages?

In general, the (oracle) test of checking whether the probability of a hypothesis exceeds the threshold value is PP-hard. Thus, PP-hardness for MPH for FO-rewritable languages cannot be avoided in general. The remaining question is whether the hypothesis can be identified efficiently, or do we really need to guess the hypothesis? Fortunately, this can be avoided, since we can walk through polynomially many hypotheses (in data complexity) and combine the threshold tests for each of the hypotheses into a single PP computation using the results of [6].

Theorem 15. *Let $\mathcal{L}$ be a Datalog$^{\pm}$ languages, for which ontology-mediated query answering is in AC^0 in data complexity. Then, $\mathsf{MPH}(\mathsf{UCQ}, \mathcal{L})$ is PP-complete in data complexity under polynomial time Turing reductions.*

Proof. PP-hardness follows from the complexity of probabilistic query evaluation over PDBs ([68], Corollary 1), since we can choose $Q = \top$ and reformulate any UCQ into a set of NCs such that the consistency of a database is equivalent to the non-satisfaction of the UCQ.

We consider an OMQ (Q, Σ), a PDB $\mathcal{P}$, and a threshold p. Since the query is FO-rewritable, it is equivalent to an ordinary UCQ Q_Σ over $\mathcal{P}$. Similarly, we can rewrite the UCQ $Q_\bot$ expressing the non-satisfaction of the NCs into a UCQ $Q_{\bot,\Sigma}$. By the observation in Theorem 5, we can enumerate all hypotheses $\mathcal{H}$, which are the polynomially many matches for Q_Σ in $\mathcal{P}$, and then have to check for each $\mathcal{H}$ whether the probability of all consistent extensions exceeds p. The latter part is equivalent to evaluating $\neg Q_{\bot,\Sigma} \wedge \bigwedge_{t \in \mathcal{H}} t$ over $\mathcal{P}$, which can be done by a PP oracle. We accept if and only if one of these PP checks yields a positive answer. In the terminology of [6], this is a polynomial-time disjunctive reduction of our problem to a PP problem. Since that paper shows that PP is closed under such reductions, we obtain the desired PP upper bound. □

Given Theorem 15, one may wonder whether the PP upper bound for MPH also applies to languages where OMQA is P-complete in data complexity. Interestingly, $\mathsf{MPH}(\mathsf{UCQ}, \mathsf{GF})$ is already $\mathrm{NP}^{\mathrm{PP}}$-hard.

Theorem 16. $\mathsf{MPH}(\mathsf{UCQ}, \mathsf{GF})$ *is $\mathrm{NP}^{\mathrm{PP}}$-hard in data complexity.*

Proof. We reduce the following problem from [74]: decide the validity of

$$\Phi = \exists x_1, \ldots, x_n \; \mathsf{C}^c \, y_1, \ldots, y_m \, \varphi,$$

where $\varphi = \varphi_1 \wedge \cdots \wedge \varphi_k$ is a propositional formula in CNF, defined over the variables $x_1, \ldots, x_n, y_1, \ldots, y_m$. This amounts to checking whether there is a partial assignment for $x_1, \ldots, x_n$ that admits at least c extensions to $y_1, \ldots, y_m$ that satisfy φ.

We can assume without loss of generality that each of the clauses φ_i contains exactly three literals: shorter clauses can be padded by copying existing literals, and longer clauses can be abbreviated using auxiliary variables that are included under the counting quantifier C^c. Since the values of these variables are uniquely determined by the original variables, this does not change the number of satisfying assignments.

We define the PDB $\mathcal{P}_\Phi$ that describes the structure of Φ:

- For each variable v occurring in Φ, $\mathcal{P}_\Phi$ contains the tuples $\langle \mathsf{V}(v, 0) : 0.5 \rangle$ and $\langle \mathsf{V}(v, 1) : 0.5 \rangle$, where v is viewed as a constant. These tuples represent the assignments that map v to *false* and *true*, respectively.
- For each clause φ_j, we introduce the tuple $\langle \mathsf{C}(v_1, t_1, v_2, t_2, v_3, t_3) : 1 \rangle$, where t_i is 1, if v_i occurs negatively in φ_j, and 0, otherwise (again, all terms are constants). For example, for $x_3 \vee \neg y_2 \vee x_7$, we use $\langle \mathsf{C}(x_3, 0, y_2, 1, x_7, 0) : 1 \rangle$. This encodes the knowledge about the partial assignments that do not satisfy φ.
- We use auxiliary atoms $\langle \mathsf{A}(x_1) : 1 \rangle$, $\langle \mathsf{S}(x_1, x_2) : 1 \rangle$, $\ldots$, $\langle \mathsf{S}(x_{n-1}, x_n) : 1 \rangle$, $\langle \mathsf{L}(x_n) : 1 \rangle$ to encode the order on the variables x_i, and similarly for y_j we use the atoms $\langle \mathsf{B}(y_1) : 1 \rangle$, $\langle \mathsf{S}(y_1, y_2) : 1 \rangle$, $\ldots$, $\langle \mathsf{S}(y_{m-1}, y_m) : 1 \rangle$, and $\langle \mathsf{L}(y_m) : 1 \rangle$.

We now describe the program Σ used for the reduction. First, we detect whether all variables x_i $(1 \leq i \leq n)$ have a truth assignment (i.e., at least one of the facts $\mathsf{V}(x_i, 0)$ or $\mathsf{V}(x_i, 1)$ is present) by the special nullary predicate A, using the auxiliary unary predicates V and A:

$$\begin{aligned} \mathsf{V}(x, t) &\to \mathsf{V}(x), \\ \mathsf{A}(x) \wedge \mathsf{V}(x) \wedge \mathsf{S}(x, x') &\to \mathsf{A}(x'), \\ \mathsf{A}(x) \wedge \mathsf{V}(x) \wedge \mathsf{L}(x) &\to \mathsf{A}, \end{aligned}$$

where x, x', t are variables. We do the same for the variables $y_1, \ldots, y_m$:

$$\begin{aligned} \mathsf{B}(y) \wedge \mathsf{V}(y) \wedge \mathsf{S}(y, y') &\to \mathsf{B}(y'), \\ \mathsf{B}(y) \wedge \mathsf{V}(y) \wedge \mathsf{L}(y) &\to \mathsf{B}. \end{aligned}$$

Now, the query $Q = \mathsf{A}$ ensures that only such hypotheses are valid that at least contain a truth assignment for the variables $x_1, \ldots, x_n$.

Next, we restrict the assignments to satisfy φ by using additional NCs in Σ. First, we ensure that there is no "inconsistent" assignment for any variable v, i.e., only one of the facts $\mathsf{V}(v, 0)$ or $\mathsf{V}(v, 1)$ holds:

$$\mathsf{V}(v, 0) \wedge \mathsf{V}(v, 1) \to \bot.$$

Furthermore, if all variables $y_1, \ldots, y_m$ have an assignment, then none of the clauses in φ can be falsified:

$$\mathsf{C}(v_1, t_1, v_2, t_2, v_3, t_3) \wedge \mathsf{V}(v_1, t_1) \wedge \mathsf{V}(v_2, t_2) \wedge \mathsf{V}(v_3, t_3) \wedge \mathsf{B} \to \bot,$$

where $v_1, t_1, v_2, t_2, v_3, t_3$ are variables. We now prove the following claim.

Claim. Φ is valid if and only if there exists a hypothesis $\mathcal{H}$ that satisfies (Q, Σ) such that all consistent databases that extend $\mathcal{H}$ sum up to a probability (under $\mathcal{P}_\Phi$) of at least $p = 0.25^n \cdot 0.25^m(3^m - 2^m + c)$.

Assume that such a hypothesis $\mathcal{H}$ exists. Since $\mathcal{H} \models (Q, \Sigma)$, we know that for each x_i $(1 \leq i \leq n)$ one of the atoms $\mathsf{V}(x_i, 0)$, $\mathsf{V}(x_i, 1)$ is included in $\mathcal{H}$. In each consistent extension of $\mathcal{H}$, it must be the case that the complementary facts (representing an inconsistent assignment for x_i) are false. In particular, these complementary facts cannot be part of $\mathcal{H}$, since then its probability would be 0. Hence, we can ignore the factor 0.25^n in the following. There are exactly $3^m - 2^m$ databases satisfying $\mathcal{H}$ that represent consistent, but incomplete assignments for the variables y_j. Since these databases do not entail B, they are all consistent, and hence counted towards the total sum. The inconsistent assignments for $y_1, \ldots, y_m$ yield inconsistent databases, which leaves us only with the 2^m databases representing proper truth assignments. Those that violate at least one clause of φ become inconsistent, and hence there are at least c such consistent databases if and only if there are at least c extensions of the assignment represented by $\mathcal{H}$ that satisfy φ. We conclude these arguments by noting that the probability of each individual choice of atoms $\mathsf{V}(y_j, t_j)$ $(1 \leq j \leq m)$ is 0.25^m.

On the other hand, if Φ is valid, then we can use the same arguments to construct a hypothesis $\mathcal{H}$ (representing the assignment for $x_1, \ldots, x_n$) that exceeds the given threshold. □

We observe a sharp contrast in data complexity results for MPH: PP vs. $\mathrm{NP}^{\mathrm{PP}}$, as summarized in Theorems 15 and 16, respectively. These results entail an interesting connection to the data complexity dichotomy for probabilistic query evaluation in PDBs [21]. Building on Theorem 15, we can show a direct reduction from MPH for FO-rewritable languages to probabilistic query evaluation in PDBs, which implies a data complexity dichotomy between P and PP [21].

Theorem 17 (dichotomy). *For FO-rewritable languages $\mathcal{L}$,* $\mathsf{MPH}(\mathsf{UCQ}, \mathcal{L})$ *is either in* P *or* PP*-hard in data complexity under polynomial time Turing reductions.*

Proof. Recall the proof of Theorem 15. According to [21], the evaluation problem for the UCQ $Q_{\bot,\Sigma}$ over a PDB $\mathcal{P}$ is either in P or PP-hard (under polynomial time Turing reductions). In the former case, MPH can also be decided in deterministic polynomial time. In the latter case, we reduce the evaluation problem for $Q_{\bot,\Sigma}$ over a PDB $\mathcal{P}$ to the MPH for Q_Σ and $Q_{\bot,\Sigma}$ over some PDB $\hat{\mathcal{P}} \supseteq \mathcal{P}$.

For the reduction, we introduce an "artificial match" for Q_Σ into $\hat{\mathcal{P}}$, by adding new constants and atoms (with probability 1) that satisfy one disjunct of Q_Σ, while taking care that these new atoms do not satisfy $Q_{\bot,\Sigma}$. Such atoms must exist if Q_Σ is not subsumed by $Q_{\bot,\Sigma}$; otherwise, all hypotheses would trivially have the probability 0 (and hence the MPH would be decidable in polynomial

time). In $\hat{\mathcal{P}}$, the probability of the most probable hypothesis for Q_Σ and $Q_{\perp,\Sigma}$ is the same as the probability of $Q_{\perp,\Sigma}$ over $\mathcal{P}$, and hence deciding the threshold is PP-hard by [21] and Corollary 1. □

5 Related Work

Research on probabilistic databases is almost as old as traditional databases as stated in [68]. We note the seminal work of [31], which has been very important in probabilistic database research as well as the first formulation of possible worlds semantics in the context of databases by [41]. Note that the possible worlds semantics has been widely employed in artificial intelligence; e.g., in probabilistic graphical models [44, 23, 54] and probabilistic logic programming [63, 57, 24, 64].

Recent advances on PDBs are mainly driven by the dichotomy result given for unions of conjunctive queries [21]. Recently, *open-world probabilistic databases* have been proposed as an alternative open-world probabilistic data model, and the data complexity dichotomy has been lifted to this open-world semantics [15, 16]. Other dichotomy results extend the dichotomy for unions of conjuncive queries in other directions; some allow for disequality ($\neq$) joins in the queries [51] and some for inequality ($<$) joins in the queries [52]. There is also a trichotomy result over queries with aggregation [59]. The common ground in all these dichotomy results is the fact that they classify queries as being safe or unsafe (while the data is not fixed). A different approach is to obtain a classification relative to the structure of the underlying database, and it has been proven in [2], for instance, that every query formulated in monadic second-order logic can be evaluated in linear time over PDBs with a bounded tree-width.

The literature on probabilistic extensions of ontology languages is rich, and we refer the interested reader to a survey [49], and focus on the most recent, and in particular, data-oriented models, which are also recently surveyed in terms of semantic expressivity [9]. Ontology-mediated queries for probabilistic databases have been investigated in the context of both description logics [43, 18] and Datalog$^\pm$ [8], and the dichotomy results from PDBs are lifted. Most of the recent work on probabilistic query answering using ontologies is based on lightweight ontology languages, such as the approaches to *Bayesian description logics* in [22, 18, 19], which combine the description logics of the *DL-Lite* family and the description logic $\mathcal{EL}$, respectively, with Bayesian networks [54]. The underlying probabilistic semantics can be generalized to other ontology languages and PGMs as well. For example, a closely related approach is the one to probabilistic Datalog$^\pm$ in [33], which combines Datalog$^\pm$ with Markov logic networks [61]. In [17], the computational complexity of query answering in probabilistic Datalog$^\pm$ under the possible worlds semantics is investigated.

Maximal posterior computational problems are inspired by PGMs [44], in particular, Bayesian networks [54]. The most probable database problem is introduced in [37], while the most probable hypothesis problem is introduced in [14]. Moreover, these problems are also studied for ontological queries in [14].

6 Conclusion

We have surveyed several query answering and reasoning tasks that can be used to exploit the full potential of probabilistic knowledge bases. In the first part of the tutorial, we focused on (tuple-independent) probabilistic databases as the simplest probabilistic data model. In the second part of the tutorial, we moved on to richer representations where the probabilistic database is extended with ontological knowledge. For each part, we surveyed some known data complexity results and highlighted some recent results.

Acknowledgments. This work was supported by The Alan Turing Institute under the UK EPSRC grant EP/N510129/1, and by the EPSRC grants EP/R013667/1, EP/L012138/1, and EP/M025268/1.

References

1. Abiteboul, S., Hull, R., Vianu, V. (eds.): Foundations of Databases: The Logical Level. Addison-Wesley Longman Publishing Co., Inc., 1st edn. (1995)
2. Amarilli, A., Bourhis, P., Senellart, P.: Tractable lineages on treelike instances: Limits and extensions. In: Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS-16). pp. 355–370. ACM (2016)
3. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, 2nd edn. (2007)
4. Baget, J.F., Mugnier, M.L., Rudolph, S., Thomazo, M.: Walking the complexity lines for generalized guarded existential rules. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11). pp. 712–717 (2011)
5. Beeri, C., Vardi, M.Y.: The implication problem for data dependencies. In: Proceedings of the 8th International Colloquium on Automata, Languages, and Programming (ICALP-81). Lecture Notes in Computer Science, vol. 115, pp. 73–85. Springer (1981)
6. Beigel, R., Reingold, N., Spielman, D.: PP is closed under intersection. Journal of Computer and System Sciences **50**(2), 191–202 (1995)
7. Bienvenu, M., Cate, B.T., Lutz, C., Wolter, F.: Ontology-based data access: A study through disjunctive Datalog, CSP, and MMSNP. ACM Transactions on Database Systems (TODS) **39**(4), 33:1–33:44 (2014)
8. Borgwardt, S., Ceylan, İ.İ., Lukasiewicz, T.: Ontology-mediated queries for probabilistic databases. In: Proceedings of the 31th AAAI Conference on Artificial Intelligence (AAAI-17). pp. 1063–1069. AAAI Press (2017)
9. Borgwardt, S., Ceylan, İ.İ., Lukasiewicz, T.: Recent advances in querying probabilistic knowledge bases. In: Lang, J. (ed.) Proceedings of the 27th International Joint Conference on Artificial Intelligence and the 23rd European Conference on Artificial Intelligence, IJCAI-ECAI 2018. IJCAI/AAAI Press (2018)
10. Calì, A., Gottlob, G., Kifer, M.: Taming the infinite chase: Query answering under expressive relational constraints. Journal of Artificial Intelligence Research **48**, 115–174 (2013)

11. Calì, A., Gottlob, G., Lukasiewicz, T.: A general Datalog-based framework for tractable query answering over ontologies. Journal of Web Semantics **14**, 57–83 (2012)
12. Calì, A., Gottlob, G., Pieris, A.: Towards more expressive ontology languages: The query answering problem. Artificial Intelligence **193**, 87–128 (2012)
13. Ceylan, İ.İ.: Query Answering in Probabilistic Data and Knowledge Bases. Ph.D. thesis, Technische Universität Dresden (2017)
14. Ceylan, İ.İ., Borgwardt, S., Lukasiewicz, T.: Most probable explanations for probabilistic database queries. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI-17). pp. 950–956. AAAI Press (2017)
15. Ceylan, İ.İ., Darwiche, A., Van den Broeck, G.: Open-world probabilistic databases. In: Proceedings of the 15th International Conference on Principles of Knowledge Representation and Reasoning (KR-16). pp. 339–348. AAAI Press (2016)
16. Ceylan, İ.İ., Darwiche, A., Van den Broeck, G.: Open-world probabilistic databases: An abridged report. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI-17). pp. 4796–4800. AAAI Press (2017)
17. Ceylan, İ.İ., Lukasiewicz, T., Peñaloza, R.: Complexity results for probabilistic Datalog±. In: Proceedings of the 28th European Conference on Artificial Intelligence (ECAI-16). Frontiers in Artificial Intelligence and Applications, vol. 285, pp. 1414–1422. IOS Press (2016)
18. Ceylan, İ.İ., Peñaloza, R.: Probabilistic query answering in the Bayesian description logic BEL. In: Proceedings of the 9th International Conference on Scalable Uncertainty Management (SUM-15). Lecture Notes in Computer Science, vol. 9310, pp. 21–35. Springer (2015)
19. Ceylan, İ.İ., Peñaloza, R.: The Bayesian ontology language $\mathcal{BEL}$. Journal of Automated Reasoning **58**(1), 67–95 (2017)
20. Dalvi, N., Suciu, D.: Efficient query evaluation on probabilistic databases. The VLDB Journal **16**(4), 523–544 (2007)
21. Dalvi, N., Suciu, D.: The dichotomy of probabilistic inference for unions of conjunctive queries. Journal of the ACM **59**(6), 1–87 (2012)
22. d'Amato, C., Fanizzi, N., Lukasiewicz, T.: Tractable reasoning with Bayesian description logics. In: Proceedings of the 2nd International Conference on Scalable Uncertainty Management (SUM-08). pp. 146–159. Springer (2008)
23. Darwiche, A.: Modeling and Reasoning with Bayesian Networks. Cambridge University Press (2009)
24. De Raedt, L., Kimmig, A., Toivonen, H.: ProbLog: A probabilistic Prolog and its application in link discovery. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07). pp. 2468–2473. Morgan Kaufmann (2007)
25. Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S., Zhang, W.: Knowledge Vault: A Web-scale approach to probabilistic knowledge fusion. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 601–610. ACM (2014)
26. Dong, X., Gabrilovich, E., Murphy, K., Dang, V., Horn, W., Lugaresi, C., Sun, S., Zhang, W.: Knowledge-based trust: Estimating the trustworthiness of Web sources. Proceedings of VLDB Endowment **8**(9), 938–949 (2015)
27. Fader, A., Soderland, S., Etzioni, O.: Identifying relations for open information extraction. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing. pp. 1535–1545. Association for Computational Linguistics (2011)

28. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: Semantics and query answering. Theoretical Computer Science **336**(1), 89–124 (2005)
29. Ferrucci, D.A.: Introduction to "This is Watson". IBM J. Res. Dev. **56**(3), 235–249 (2012)
30. Ferrucci, D., Levas, A., Bagchi, S., Gondek, D., Mueller, E.T.: Watson: Beyond Jeopardy! Artificial Intelligence **199-200**, 93–105 (2013)
31. Fuhr, N., Rölleke, T.: A probabilistic relational algebra for the integration of information retrieval and database systems. ACM Transactions on Database Systems (TOIS) **15**(1), 32–66 (1997)
32. Gill, J.T.: Computatonal complexity of probabilistic Turing machines. SIAM Journal on Computing **6**(4), 675–695 (1977)
33. Gottlob, G., Lukasiewicz, T., Martinez, M.V., Simari, G.I.: Query answering under probabilistic uncertainty in Datalog± ontologies. Annals of Mathematics and Artificial Intelligence **69**(1), 37–72 (2013)
34. Grädel, E., Gurevich, Y., Hirsch, C.: The complexity of query reliability. In: Proceedings of the 17th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS-98). pp. 227–234. ACM (1998)
35. Greenemeier, L.: Human traffickers caught on hidden internet. Scientific American (2015)
36. Gribkoff, E., Suciu, D.: SlimShot: In-database probabilistic inference for knowledge bases. Proceedings of VLDB Endowment **9**(7) (2016)
37. Gribkoff, E., Van den Broeck, G., Suciu, D.: The most probable database problem. In: Proceedings of the 1st International Workshop on Big Uncertain Data (BUDA) (2014)
38. Gribkoff, E., Van den Broeck, G., Suciu, D.: Understanding the complexity of lifted inference and asymmetric weighted model counting. In: Proceedings of the 30th Annual Conference on Uncertainty in Artificial Intelligence (UAI-14). pp. 280–289. AUAI Press (2014)
39. Hesse, W., Allender, E., Barrington, D.A.M.: Uniform constant-depth threshold circuits for division and iterated multiplication. Journal of Computer and System Sciences **65**(4), 695–716 (2002)
40. Hoffart, J., Suchanek, F.M., Berberich, K., Weikum, G.: Yago2: A spatially and temporally enhanced knowledge base from Wikipedia. Artificial Intelligence **194**, 28–61 (2013)
41. Imieliski, T., Lipski, W.: Incomplete information in relational databases. Journal of the ACM **31**(4), 761–791 (1984)
42. Jaeger, M.: Relational Bayesian networks. In: Proceedings of the 23rd Annual Conference on Uncertainty in Artificial Intelligence (UAI-97). pp. 266–273. Morgan Kaufmann (1997)
43. Jung, J.C., Lutz, C.: Ontology-based access to probabilistic data with OWL QL. In: Proceedings of the 11th International Conference on The Semantic Web—Volume Part I. pp. 182–197. Springer (2012)
44. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. MIT Press (2009)
45. Krötzsch, M., Rudolph, S.: Extending decidable existential rules by joining acyclicity and guardedness. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11). pp. 963–968. AAAI Press (2011)
46. Ku, J.P., Hicks, J.L., Hastie, T., Leskovec, J., Ré, C., Delp, S.L.: The mobilize center: An NIH big data to knowledge center to advance human movement research and improve mobility. Journal of the American Medical Informatics Association **22**(6), 1120–1125 (2015)

47. Libkin, L.: Elements of Finite Model Theory. Springer (August 2004)
48. Littman, M.L., Goldsmith, J., Mundhenk, M.: The computational complexity of probabilistic planning. Journal of Artificial Intelligence Research **9**, 1–36 (1998)
49. Lukasiewicz, T., Straccia, U.: Managing uncertainty and vagueness in description logics for the Semantic Web. Journal of Web Semantics **6**(4), 291–308 (2008)
50. Mitchell, T., Cohen, W., Hruschka, E., Talukdar, P., Betteridge, J., Carlson, A., Dalvi, B., Gardner, M., Kisiel, B., Krishnamurthy, J., Lao, N., Mazaitis, K., Mohamed, T., Nakashole, N., Platanios, E., Ritter, A., Samadi, M., Settles, B., Wang, R., Wijaya, D., Gupta, A., Chen, X., Saparov, A., Greaves, M., Welling, J.: Never-ending learning. In: Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI-15). pp. 2302–2310 (2015)
51. Olteanu, D., Huang, J.: Using OBDDs for efficient query evaluation on probabilistic databases. In: Proceedings of the 2nd International Conference on Scalable Uncertainty Management (SUM-08). Lecture Notes in Computer Science, vol. 5291, pp. 326–340 (2008)
52. Olteanu, D., Huang, J.: Secondary-storage confidence computation for conjunctive queries with inequalities. In: Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data. pp. 389–402. ACM (2009)
53. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1994)
54. Pearl, J.: Probabilistic Reasoning in Intelligent Systems. Morgan Kaufmann (1988)
55. Peters, S.E., Zhang, C., Livny, M., Ré, C.: A machine reading system for assembling synthetic paleontological databases. PLoS ONE **9**(12) (2014)
56. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. Journal on Data Semantics **10** (2008)
57. Poole, D.: The independent choice logic for modelling multiple agents under uncertainty. Artificial Intelligence **94**(1-2), 7–56 (1997)
58. Provan, J.S., Ball, M.O.: The complexity of counting cuts and of computing the probability that a graph is connected. SIAM Journal on Computing **12**(4), 777–788 (1983)
59. Ré, C., Suciu, D.: The trichotomy of having queries on a probabilistic database. The VLDB Journal **18**(5), 1091–1116 (2009)
60. Reiter, R.: On closed world data bases. Logic and Data Bases pp. 55–76 (1978)
61. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning **62**(1), 107–136 (2006)
62. Rossman, B.: Homomorphism preservation theorems. Journal of the ACM **55**(3), 1–53 (2008)
63. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: Proceedings of the 12th International Conference on Logic Programming (ICLP-95). pp. 715–729. MIT Press (1995)
64. Sato, T., Kameya, Y.: PRISM: A language for symbolic-statistical modeling. In: Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97). pp. 1330–1335. Morgan Kaufmann (1997)
65. Shin, J., Wu, S., Wang, F., De Sa, C., Zhang, C., Ré, C.: Incremental knowledge base construction using DeepDive. Proceedings of VLDB Endowment **8**(11), 1310–1321 (2015)
66. Sipser, M.: Introduction to the Theory of Computation. International Thomson Publishing, 1st edn. (1996)
67. Staworko, S., Chomicki, J.: Consistent query answers in the presence of universal constraints. Inf. Syst. **35**(1), 1–22 (2010)
68. Suciu, D., Olteanu, D., Ré, C., Koch, C.: Probabilistic Databases. Synthesis Lectures on Data Management, Morgan & Claypool Publishers (2011)

69. Toda, S.: On the computational power of PP and ⊕P. In: Proceedings of the 30th Annual Symposium on Foundations of Computer Science. pp. 514–519 (1989)
70. Toda, S., Watanabe, O.: Polynomial-time 1-Turing reductions from #PH to #P. Theoretical Computer Science **100**(1), 205–221 (1992)
71. Valiant, L.G.: The complexity of computing the permanent. Theoretical Computer Science **8**(2), 189–201 (1979)
72. Vardi, M.Y.: The complexity of relational query languages. In: Lewis, H.R., Simons, B.B., Burkhard, W.A., Landweber, L.H. (eds.) Proceedings of the 14th Annual ACM Symposium on Theory of Computing (STOC-82). pp. 137–146. ACM (1982)
73. Vollmer, H.: Introduction to Circuit Complexity: A Uniform Approach. Springer (1999)
74. Wagner, K.W.: The complexity of combinatorial problems with succinct input representation. Acta Informatica **23**(3), 325–356 (1986)
75. Wu, W., Li, H., Wang, H., Zhu, K.Q.: Probase: A probabilistic taxonomy for text understanding. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. pp. 481–492. ACM (2012)