Open-World Probabilistic Databases: Semantics, Algorithms, Complexity

İsmail İlkan Ceylan[a,*], Adnan Darwiche[b], Guy Van den Broeck[b]

[a] *Department of Computer Science, University of Oxford, Wolfson Building, Parks Road, Oxford OX1 3QD, UK*
[b] *Computer Science Department, University of California, Los Angeles, 404 Westwood Plaza, Los Angeles, CA 90095, US*

Abstract

Large-scale probabilistic knowledge bases are becoming increasingly important in academia and industry. They are continuously extended with new data, powered by modern information extraction tools that associate probabilities with knowledge base facts. The state of the art to store and process such data is founded on probabilistic databases. Many systems based on probabilistic databases, however, still have certain semantic deficiencies, which limit their potential applications. We revisit the semantics of probabilistic databases, and argue that the *closed-world assumption* of probabilistic databases, i.e., the assumption that facts not appearing in the database have the probability *zero*, conflicts with the everyday use of large-scale probabilistic knowledge bases. To address this discrepancy, we propose *open-world probabilistic databases*, as a new probabilistic data model. In this new data model, the probabilities of unknown facts, also called *open facts*, can be assigned any probability value from a default probability interval. Our analysis entails that our model aligns better with many real-world tasks such as *query answering*, *relational learning*, *knowledge base completion*, and *rule mining*. We make various technical contributions. We show that the *data complexity dichotomy*, between polynomial time and #P, for evaluating unions of conjunctive queries on probabilistic databases can be lifted to our open-world model. This result is supported by an algorithm that computes the probabilities of the so-called *safe* queries efficiently. Based on this algorithm, we prove that evaluating safe queries is in *linear time* for probabilistic databases, under reasonable assumptions. This remains true in open-world probabilistic databases for a more restricted class of safe queries. We extend our data complexity analysis beyond unions of conjunctive queries, and obtain a host of complexity results for both classical and open-world probabilistic databases. We conclude our analysis with an in-depth investigation of the *combined complexity* in the respective models.

Keywords: knowledge bases, probabilistic databases, semantics, closed-world assumption, open-world assumption, inference, credal sets, learning, data complexity, dichotomy, lifted inference

1. Introduction

Driven by the need to learn from vast amount of text data, efforts throughout *information extraction*, *natural language processing (e.g., question answering)*, *relational learning*, *knowledge representation and reasoning*, and *databases* are coming together to build *large-scale knowledge bases* and reason over them. Academic systems such as NELL [1], DeepDive [2], Reverb [3], and YAGO [4] continuously crawl the Web to extract *structured information*. Industry projects such as Microsoft's Probase [5], or Google's Knowledge Vault [6] similarly learn structured data from text, and thus populate their databases with millions of entities and billions of facts. Thus, research on large-scale knowledge bases serves as an important frontier for *artificial intelligence (AI)*.

Systems such as DeepDive have been used by scientists to build knowledge bases of *gene interactions*, *paleobiology*, and *geoscience*, all by reading scientific publications and extracting structured knowledge [7, 8]. One of the most visible applications of these probabilistic knowledge bases is in *search engines* (see, e.g., Google search results), i.e.,

*Corresponding author.
Email addresses: ismail.ceylan@cs.ox.ac.uk (İsmail İlkan Ceylan), darwiche@cs.ucla.edu (Adnan Darwiche), guyvdb@cs.ucla.edu (Guy Van den Broeck)

the standard list of relevant web pages is often augmented with a table of structured data that pertains to the search query, which is clearly linked to the underlying knowledge base.

Knowledge bases contain data which is necessarily uncertain. To go from the raw text to structured data, information extraction systems employ a sequence of statistical machine learning techniques, from part-of-speech tagging until relation extraction [9]. For knowledge-base completion – the task of deriving new facts from existing knowledge – statistical relational learning algorithms make use of embeddings [10, 11] or probabilistic rules [12, 13]. In both settings, the output is a predicted fact with a probability, or confidence, value. It is therefore common to interpret such knowledge through a probabilistic semantics.

The classical and most basic model to represent probabilistic data is that of *tuple-independent probabilistic databases* (PDBs) [14], which indeed underlies some of these systems [6, 2]. Probabilistic databases, however, lack a suitable handling of incompleteness. In particular, each of the above systems encodes only a portion of the real-world, and this description is necessarily *incomplete*. For example, according to YAGO, the average number of children per person is 0.02 [15]. However, when it comes to querying, most of these systems employ the *closed-world assumption* (CWA) [16] – according to the tuple-independent PDB semantics, each database atom is an independent Bernoulli random variable, and all other atoms have probability zero. That is, many facts are assumed to be impossible, although they actually have some unknown probability in $[0, 1]$.

In this paper, we revisit the CWA of probabilistic databases, and observe that the *CWA is violated* in the deployment of these systems, which makes it *problematic to reason, learn, or mine* on top of these databases. We will argue the following salient points in detail. First, query answering under the CWA does not take into account the effect the open-world can have on the query probability. This makes it impossible to distinguish queries whose probability should intuitively differ. Second, knowledge bases are part of a larger machine learning loop that continuously updates beliefs about facts based on new evidence. From a Bayesian learning perspective [17], this loop can only be principled when learned facts have an a priori *non-zero* probability. The CWA does not accurately represent this mode of operation and puts it on weak footing. Third, the CWA is problematic for higher level tasks that one is usually interested in performing on probabilistic databases, including some principled approaches to knowledge base completion and mining. Finally, we note that these issues are not temporary: it will never be possible to complete probabilistic knowledge bases of even the most trivial relations, as the memory requirements quickly become excessive. This already manifests itself today: statistical classifiers output facts at a high rate, but only the most probable ones make it into the knowledge base, and the rest is truncated, losing much of the statistical information. For example, 99% of the facts in NELL have a probability larger than 0.91.

We propose a new semantics for probabilistic knowledge bases to address these problems, based on the *open-world assumption (OWA)*. In contrast to the CWA, the OWA does not presume that the knowledge of a domain is complete, and as a consequence, all open atoms remain possible. Our proposal for *open-world probabilistic databases* (OpenPDBs) builds on the theory of *imprecise probabilities*, and *credal sets* [18], to allow interval-based probabilities for open atoms. OpenPDBs define a probability threshold to determine which facts make it into the knowledge base, which is motivated by the mode of operation in systems that learn knowledge bases. In OpenPDBs, all facts in the open world must have a lower probability, which bounds their contribution to the probability of possible worlds. This data model provides more meaningful answers, in terms of *upper* and *lower* bounds on the query probability. Throughout this paper, we assume a *finite* domain, but we include a discussion of probabilistic reasoning with a possibly infinite number of objects as well as other recent extensions in the related work section.

The organization of this paper is as follows. Section 2 is dedicated to preliminaries, where we provide an overview on logics, databases, and the query languages that are relevant to our study. This section also includes a brief background on some complexity classes which are central to this paper. In Section 3, we recall (tuple-independent) probabilistic databases and analyze the CWA, and its implications in practice, based on the above-mentioned desiderata. In Section 4, we introduce OpenPDBs, and discuss how this model evaluates in practice. The decision problems regarding probabilistic query evaluation are introduced in Section 5 with an overview of existing results. Section 6 contains all the *data complexity* results, and Section 7 all the *combined complexity* results obtained in this paper. We review the related work in Section 8, and locate our approach in the existing literature. We conclude with discussions for future work and concluding remarks. For presentation purposes, we defer the proofs to the appendix of this paper.

This work is based on a conference paper which appeared first in KR 2016 [19], and later also as an abridged report in IJCAI 2017 [20]. This paper extends the conference version with all technical preliminaries and proof details. We also provide a complete picture for our complexity landscape, by including an analysis on the combined complexity.

Most of these results appeared earlier, as part of the dissertation of the first author [21].

2. Preliminaries

We recall first-order logic and databases with a special focus on the query answering problem, and various query languages. We conclude by providing some complexity background that is relevant for this paper.

2.1. Logic and Notation

We focus on the *function-free* fragment of first-order logic (FOL) and assume a *finite domain*. A relational vocabulary σ consists of finite sets $\mathbf{R}$ of *predicates*, $\mathbf{C}$ of *constants*, and a (possibly infinite) set $\mathbf{V}$ of *variables*. The function $\mathsf{ar} : \mathbf{R} \mapsto \mathbb{N}$ associates a natural number to each predicate $\mathsf{R} \in \mathbf{R}$ that defines the (unique) arity of R. A *term* is either a constant or a variable. An *atom* is of the form $\mathsf{R}(s_1, \ldots, s_n)$, where R is an n-ary predicate, and $s_1, \ldots, s_n$ are terms. A *ground atom* (also called *fact*, *record*, or *tuple*) is an atom that contains only constants as terms.

First-order formulas are built from atoms inductively via *negation*, *conjunction*, *disjunction*, *existential quantification*, and *universal quantification* as usual, using the syntax rule:

$$\Phi = \mathsf{R}(s_1, \ldots, s_n) \;\mid\; \neg\Phi \;\mid\; \Phi \wedge \Phi \;\mid\; \Phi \vee \Phi \;\mid\; \exists x.\Phi \;\mid\; \forall x.\Phi,$$

where $\mathsf{R}(s_1, \ldots, s_n)$ is an atom, and x is a variable. We express *implication* $\Phi \rightarrow \Psi = \neg\Phi \vee \Psi$; *truth* $\top = \Phi \vee \neg\Phi$; and *falsity* $\bot = \Phi \wedge \neg\Phi$, as usual.

A formula is *quantifier-free* if it 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*. We use a vector notation to denote a sequence of variables $x_1, \ldots, x_n$ by $\vec{x}$, and use $\Phi(\vec{x})$ to represent a formula Φ with free variables $\vec{x}$. A *(first-order) sentence* is a first-order formula without any free variables, also called a *closed formula*. A formula is *positive (or monotone)* if it does not contain negations, but can contain the truth constant $\top$. A *literal* is either an atom or its negation. A *disjunctive clause* is a disjunction of literals and a *conjunctive clause* is a conjunction of literals.

The semantics of first-order logic over finite domains can be defined in terms of Herbrand interpretations [22]. The *Herbrand base* of a relational vocabulary σ is the set of all ground atoms that can be constructed from the set of predicates ($\mathbf{R}$) and set of constants ($\mathbf{C}$). An *interpretation* is then a truth-value assignment to all the ground atoms in the Herbrand base. An interpretation ω is a model of formula Φ, denoted by $\omega \models \Phi$, if ω satisfies Φ, defined in the usual way.

Let FO be the class of first-order formulas. The class of *existential first-order formulas* ($\exists$FO) consists of first-order formulas of the form $\exists\vec{x}.\Phi(\vec{x})$; the class of *universal first-order formulas* ($\forall$FO) consists of first-order formulas of the form $\forall\vec{x}.\Phi(\vec{x})$, where Φ is any Boolean combination of atoms. The class of formulas in *existential conjunctive normal form* ($\exists$CNF) consists of first-order formulas of the form $\exists\vec{x}.\Phi(\vec{x})$; the class of formulas in *universal conjunctive normal form* ($\forall$CNF) consists of first-order formulas of the form $\forall\vec{x}.\Phi(\vec{x})$, where in both cases, Φ is a conjunction of disjunctive clauses. The class of formulas in *existential disjunctive normal form* ($\exists$DNF) consists of formulas of the form $\exists\vec{x}.\Phi(\vec{x})$; the class of formulas in *universal disjunctive normal form* ($\forall$DNF) consists of formulas of the form $\forall\vec{x}.\Phi(\vec{x})$, where in both cases, Φ is a disjunction of conjunctive clauses. The class of formulas in *conjunctive normal form* (CNF) consists of $\exists$CNF and $\forall$CNF formulas. The class of formulas in *disjunctive normal form* (DNF) consists of $\exists$DNF and $\forall$DNF formulas. We also write kCNF, or kDNF, to denote the class of formulas, where k is the maximal number of atoms that a clause can contain.

2.2. Databases and Query Languages

Relational databases [23] are standard tools for data management. They provide sophisticated means for *storing*, *processing*, and *querying* data sources. Intellectual roots of database theory are in first-order logic [24]; in particular, in finite model theory [25]. We thus adopt a *model-theoretic* perspective and view a database as a Herbrand interpretation and denote it by $\mathcal{D}$.

A classical representation of a relational database is in terms of database tables, which organize atoms relative to the relations. Each table corresponds to a *predicate* and its rows correspond to *ground atoms* of that predicate, which are also called *records*, *facts*, or *tuples*. For example, Table 1 consists of two relational database tables. The atoms

Table 1: The relational database $\mathcal{D}_m$ represented in terms of relational database tables. Each row is interpreted as a ground atom. For example, the first row in the left table is interpreted as StarredIn(will_smith, ali).

StarredIn	
will_smith	ali
arquette	scream
pitt	mr_ms_smith
jolie	mr_ms_smith

Couple	
arquette	cox
pitt	jolie
pitt	aniston
kunis	kutcher

that appear in a table are mapped to *true*, while ones not listed in any of these tables are mapped to *false,* according to the CWA [16]. Similarly, a database $\mathcal{D}$ is sometimes represented as a set that contains all ground atoms mapped to *true*. For instance, the database from Table 1 is given as:

$$\mathcal{D}_m = \{\mathsf{StarredIn}(\mathsf{will\_smith}, \mathsf{ali}), \mathsf{StarredIn}(\mathsf{arquette}, \mathsf{scream}), \ldots, \mathsf{Couple}(\mathsf{kunis}, \mathsf{kutcher})\}.$$

The most fundamental task in databases is *query answering*; that is, given a database $\mathcal{D}$ and a query, i.e., a formula $\Phi(x_1, \ldots, x_n)$ of first-order logic, to find all substitutions (answers) $[x_1/a_1, \ldots, x_n/a_n]$ for free variables such that $\mathcal{D} \models \Phi[x_1/a_1, \ldots, x_n/a_n]$. We focus on the special case called *(Boolean) query evaluation*, that is, given a database $\mathcal{D}$ and a *closed* first-order formula Φ, 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 recall. A *conjunctive query* (CQ) over σ is an existentially quantified formula $\exists x_1 \ldots x_n.\phi$, where ϕ is a conjunction of atoms constructed from vocabulary σ. A *union of conjunctive queries* (UCQs) is a disjunction of conjunctive queries over the same free variables. Consider, for example, the query:

$$Q_1(x, y) = \exists z\, \mathsf{StarredIn}(x, z) \wedge \mathsf{StarredIn}(y, z) \wedge \mathsf{Couple}(x, y),$$

which asks for couples that starred in the same movie. This formula is an existentially quantified conjunction of atoms, i.e., a conjunctive query. Following common convention, we sometimes write the atoms as a comma-separated list, and drop the existential quantifiers:

$$Q_1(x, y) = \mathsf{StarredIn}(x, z), \mathsf{StarredIn}(y, z), \mathsf{Couple}(x, y).$$

Answers to such queries are tuples of constants from the database that match the query. For example, $\mathcal{D}_m$ has only one answer to $Q_1(x, y)$, i.e., $[x/\mathsf{pitt}, y/\mathsf{jolie}]$. Throughout the paper, we focus on *Boolean* queries, and on the query evaluation problem. Answers to such queries are either *true* or *false*. For example, the query:

$$Q_2 = \exists x, y, z\, \mathsf{StarredIn}(x, z), \mathsf{StarredIn}(y, z), \mathsf{Couple}(x, y),$$

returns *true* on the database $\mathcal{D}_m$ since there is a match for the query, i.e., $[x/\mathsf{pitt}, y/\mathsf{jolie}, z/\mathsf{mr\_ms\_smith}]$.

Note that the class UCQ corresponds to *positive* $\exists$DNF sentences. We also refer to unions of $\forall$CNF sentences, denoted UCNF, which is a disjunction of $\forall$CNF sentences. For example, the query:

$$Q_{\forall\mathrm{CNF}} = \forall x, y, z\, (\mathsf{Actor}(x) \vee \mathsf{StarredIn}(y, z)) \wedge (\mathsf{StarredIn}(x, y) \vee \mathsf{Male}(x)),$$

is in $\forall$CNF and can be rewritten into UCNF as:

$$Q_{\mathrm{UCNF}} = \forall x, y\, \mathsf{Actor}(x) \wedge (\mathsf{StarredIn}(x, y) \vee \mathsf{Male}(x)) \vee \forall x, y, z\, \mathsf{StarredIn}(y, z) \wedge (\mathsf{StarredIn}(x, y) \vee \mathsf{Male}(x)).$$

This concludes our overview on databases and query languages.

2.3. Complexity Background

We assume familiarity with the basics of complexity theory [26], and introduce the complexity classes that are most relevant to the presented results.

FP is defined as the class of functions $f : \{0,1\}^* \to \{0,1\}^*$ computable by a polynomial-time deterministic Turing Machine. The function class #P is central for problems related to counting [27], and contains the computation problems that can be expressed as the number of accepting paths of a nondeterministic polynomial-time Turing machine. The canonical problem for #P is #SAT, that is, given a propositional formula ϕ, the task of computing the number of satisfying assignments to ϕ.

In this paper, we focus on decision problems and the associated complexity classes. Intuitively, the complexity class PP [28] 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. Many problems in the AI literature, e.g., decisions about probabilistic inference in Bayesian networks, are PP-complete [29].

PP is closed under *truth table reductions* [30]; in particular, this implies that PP is closed under *complement*, *union*, and *intersection*. Due to Toda's celebrated result, it is known that $\mathrm{PH} \subseteq \mathrm{P}^{\#\mathrm{P}}$; that is, a polynomial time deterministic Turing machine with access to a #P oracle is capable of deciding all problems in the polynomial hierarchy [31]. The close connection between PP and #P is also given by Toda's theorem, which proves $\mathrm{P}^{\mathrm{PP}} = \mathrm{P}^{\#\mathrm{P}}$ [32].

Other classes of interest are $\mathrm{NP}^{\mathrm{PP}}$ and $\mathrm{PP}^{\mathrm{NP}}$, which intuitively combine search and optimization problems. A canonical problem for $\mathrm{PP}^{\mathrm{NP}}$ can be obtained by extending MAJSAT with quantified formulas [33]. A natural canonical problem for $\mathrm{NP}^{\mathrm{PP}}$ is EMAJSAT [34]; that is, given a propositional formula ϕ and a set of distinguished variables $\vec{x}$ from ϕ, is there an assignment μ to $\vec{x}$-variables such that majority of the assignments τ that extend μ satisfies ϕ. The class $\mathrm{NP}^{\mathrm{PP}}$ is important for probabilistic inference, and planning tasks. For instance, *maximum a posteriori probability (MAP)* inference in Bayesian Networks is $\mathrm{NP}^{\mathrm{PP}}$-complete [35]. The relation of these complexity classes to other classes can be summarized as follows:

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PP} \subseteq \mathrm{PP}^{\mathrm{NP}} \subseteq \mathrm{P}^{\mathrm{PP}} = \mathrm{P}^{\#\mathrm{P}} \subseteq \mathrm{NP}^{\mathrm{PP}} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXP}.$$

For decision classes, many-one reductions are standard. On the other hand, for classes, such as #P, different types of reductions are widely used. The most common reductions for #P are the so-called *polynomial time Turing reductions*, also known as Cook reductions [36]. All of our results in this paper are given under standard many-one reductions, except the dichotomy results, which are given under polynomial time Turing reductions. Hence, we will implicitly assume many-one reductions, unless explicitly stated otherwise.

When analyzing the complexity of query evaluation, our main focus is on *data complexity* which is calculated only based on the size of the database, i.e., the query is assumed to be fixed, as usual [37]. The *combined complexity* of query evaluation is calculated by considering all the components, i.e., the database, and the query, as part of the input. We also study *(bounded-arity) combined complexity* (or, simply bounded-arity complexity) which assumes that the maximum arity of the predicates is bounded by an integer constant. Note that both data and combined complexity are fairly standard in database theory. We follow standard conventions for hardness and completeness of problems in data and combined complexity (Chapter 6, [25]).

3. Probabilistic Databases

The literature on probabilistic databases is rich and there are many different types of probabilistic data models; for details, see e.g. [14, 38]. We adopt the simplest probabilistic database model, which is based on the *tuple-independence assumption*. Despite its limitations, the tuple-independent PDB model is very powerful. For instance, inference in Markov Logic Networks can be reduced to query evaluation in PDBs [39], and analogous results are known for a restricted class of ontology languages [40]. Tuple-independent probabilistic databases generalize classical databases by associating every database atom with a probability value.

Definition 3.1. 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 ground atom over σ 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 2: The PDB $\mathcal{P}_m = \{\langle \mathsf{StarredIn}(\mathsf{will\_smith}, \mathsf{ali}) : 0.9\rangle, \ldots, \langle \mathsf{Couple}(\mathsf{kunis}, \mathsf{kutcher}) : 0.7\rangle\}$ represented in terms of database tables. Each row is interpreted as a probabilistic atom $\langle t : p\rangle$, where t is a (ground) atom and p represents its probability.

StarredIn		P
will_smith	ali	0.9
will_smith	sharktale	0.8
jada_smith	ali	0.6
arquette	scream	0.7
pitt	mr_ms_smith	0.5
jolie	mr_ms_smith	0.7
jolie	sharktale	0.9

Couple		P
arquette	cox	0.6
pitt	jolie	0.8
thornton	jolie	0.6
pitt	aniston	0.9
kunis	kutcher	0.7

Table 2 shows an example PDB where each row in a table represents an atom, and each atom is now also associated with a probability value. A PDB can be viewed as a factored representation of exponentially many *possible worlds* (databases), each of which has a probability of being the true world. Both in the AI [41, 42, 43] and database literature [14], this is known as the *possible world semantics*. In PDBs, each database atom is viewed as an independent Bernoulli random variable, by the *tuple-independence assumption*. Each *world* is then simply a *classical database*, which sets a choice for all database atoms in the PDB. The CWA forces all atoms that are not present in the database to have probability zero.

Definition 3.2. A PDB $\mathcal{P}$ for vocabulary σ induces a *unique probability distribution* $\mathrm{P}_{\mathcal{P}}$ over 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 tuple 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 facts missing from PDB $\mathcal{P}$ is a *probabilistic counterpart* of the CWA. Let us now illustrate the semantics of PDBs on a simple example.

Example 3.3. Consider the PDB $\mathcal{P}_m$ from Table 2 and the database:

$$\mathcal{D}_m = \{\mathsf{StarredIn}(\mathsf{will\_smith}, \mathsf{ali}), \ldots, \mathsf{Couple}(\mathsf{kunis}, \mathsf{kutcher})\},$$

as given in Table 1. The probability of the world $\mathcal{D}_m$ can then be computed as follows:

$$\mathrm{P}(\mathcal{D}_m) = 0.9 \cdot (1 - 0.8) \cdots (1 - 0.6) \cdot 0.9 \cdot 0.7.$$

If we further add the fact $\mathsf{Couple}(\mathsf{will\_smith}, \mathsf{aniston})$ to $\mathcal{D}_m$, then $\mathrm{P}(\mathcal{D}_m) = 0$ because the additional fact does not appear in the PDB $\mathcal{P}_m$.

Queries are interpreted through the possible world semantics, which amounts to walking through all the possible worlds, and summing the probabilities of those worlds that satisfy the query.

Definition 3.4 (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 this makes probabilistic query evaluation a computationally very demanding task, but as we discuss later, in certain cases, computing the query probability efficiently is known to be feasible.

Consider for instance the PDB $\mathcal{P}_m$, and the query $Q = \exists x, y, z\ \mathsf{StarredIn}(x, y), \mathsf{Couple}(x, z)$. To evaluate Q, we can naïvely check, for each world $\mathcal{D}$, whether $\mathcal{D} \models Q$, and sum over the probabilities of the worlds, for which the satisfaction relation holds. However, it is *easy* to compute the probability of the above query in a different way. Notably, this is the case for *any* PDB and not only for our toy PDB in data complexity. We will later look into existing results on query evaluation in order to provide more insights on easy, and hard queries. We now evaluate the probabilistic database model and its use in practice, which constitutes our main motivation.

3.1. Probabilistic Databases in Practice

We evaluate PDBs on several criteria, and illustrate certain limitations, which are inherent to the semantics of PDBs and the widely employed assumptions. Most importantly, the CWA presumes complete knowledge about the domain being represented, and this assumption is warranted in many cases [16]. For example, *when a flight does not appear in an airline database*, we can be sure that *it never took place*. In what follows, we assess the adequacy of the CWA for probabilistic databases.

3.1.1. Distinguishing Queries in PDBs

The fact that many queries evaluate to probability zero makes it impossible to distinguish a large class of queries, which should *intuitively* differ, as we illustrate next.

Example 3.5 (specificity)**.** Consider the PDB $\mathcal{P}_m$ from Table 2 and the following queries:

$$Q_1(x, y) = \exists z\, \mathsf{StarredIn}(x, z) \wedge \mathsf{StarredIn}(y, z) \wedge \mathsf{Couple}(x, y),$$
$$Q_2 = \exists x, y, z\, \mathsf{StarredIn}(x, z) \wedge \mathsf{StarredIn}(y, z) \wedge \mathsf{Couple}(x, y).$$

Let us consider the queries $Q_1(\mathsf{pitt}, \mathsf{jolie})$ and Q_2. From a logical perspective, $Q_1(\mathsf{pitt}, \mathsf{jolie})$ entails Q_2, i.e., $Q_1(\mathsf{pitt}, \mathsf{jolie}) \models Q_2$. In other words, the pattern specified by $Q_1(\mathsf{pitt}, \mathsf{jolie})$ is only a special case of the pattern specified by Q_2. Hence, the reasonable expectation in an open-world setting is that $\mathrm{P}(Q_2)$ is most likely to have a larger probability than $\mathrm{P}(Q_1(\mathsf{pitt}, \mathsf{jolie}))$, since there exist a large number of couples, for which we do not yet have information, and that could satisfy the query Q_2. Under the CWA, however, $\mathrm{P}(Q_2) = \mathrm{P}(Q_1(\mathsf{pitt}, \mathsf{jolie})) = 0.28$ in the PDB $\mathcal{P}_m$.

Example 3.5 shows that query semantics under the CWA fails to distinguish a *query* from a particular *instance of this query*. In our next example, we consider two logically *incomparable queries* that have varying level of *support* in the database.

Example 3.6 (support)**.** Consider the queries $Q_1(\mathsf{will\_smith}, \mathsf{jada\_smith})$ and $Q_1(\mathsf{thornton}, \mathsf{aniston})$. The former query is *supported* by two facts in the PDB $\mathcal{P}_m$ (both people have starred in the same movie), while the latter query is supported by none, which should make it less likely. Conversely, the number of tuples to be added to the PDB $\mathcal{P}_m$ to satisfy $Q_1(\mathsf{thornton}, \mathsf{aniston})$ are more than the number of tuples to be added to the PDB $\mathcal{P}_m$ to satisfy $Q_1(\mathsf{will\_smith}, \mathsf{jada\_smith})$. Observe, however, that

$$\mathrm{P}(Q_1(\mathsf{thornton}, \mathsf{aniston})) = \mathrm{P}(Q_1(\mathsf{will\_smith}, \mathsf{jada\_smith})) = 0,$$

that is, both of the queries evaluate to probability zero under the CWA.

Example 3.6 shows that queries that do not have a matching answer in the database are viewed to be the same by the query semantics, even though these queries clearly have different levels of support in the database. The following example takes these observations to the *extreme*, by comparing the probabilities of a *satisfiable query* with an *unsatisfiable query*.

Example 3.7 (satisfiability)**.** The query $\mathsf{StarredIn}(x, y) \wedge \neg\mathsf{StarredIn}(x, y)$ is an *unsatisfiable query*, whose probability is zero on any database. The query $Q_1(\mathsf{will\_smith}, \mathsf{jada\_smith})$, on the other hand, is a *satisfiable query*, but evaluates to the same probability (i.e., zero) on the PDB $\mathcal{P}_m$.

The CWA forces a strict view on query probability, and as a result, it is not always possible to distinguish a satisfiable query from an unsatisfiable one, by comparing their probabilities.

3.1.2. Learning, Mining, and Knowledge Base Completion in PDBs

We analyze the consequences of the CWA in the context of higher-level tasks that one is usually interested in performing on probabilistic databases. These tasks range from learning, or mining to tasks such as knowledge base completion, as we detail in the sequel.

Let us consider the Bayesian learning paradigm, which is a popular view on machine learning, where the learner maintains beliefs about the world as a probability distribution, and updates these beliefs based on data, to obtain a posterior distribution. Probabilistic data (and knowledge) bases can be cast into this principled framework, as follows.

Suppose that we are building a probabilistic knowledge base from scratch. The first step of Bayesian learning is to come up with a prior belief about the facts in the database. Next, as we read the web, we incorporate more evidence into our distribution. For example, suppose we observe two web pages, d_a and d_b, and are interested in querying for Q_2 as defined above. Then, we may have

$$\mathrm{P}(Q_2) = 0.01, \qquad \mathrm{P}(Q_2 \mid d_a) = 0.09, \qquad \mathrm{P}(Q_2 \mid d_a, d_b) = 0.08,$$

that is, our prior belief is that the probability of Q_2 is 1%, but after observing the information on web page d_a, that probability becomes 9%. When additionally observing web page d_b, giving evidence to the contrary, the belief drops to 8%.

This sequence is a typical run of Bayesian learning. Unfortunately, it is not the mode of operation for large-scale PDBs as they currently function. A typical run would instead be

$$\mathrm{P}(Q_2) = 0, \qquad \mathrm{P}(Q_2 \mid d_a) = 0.09, \qquad \mathrm{P}(Q_2 \mid d_a, d_b) = 0.08,$$

The difference is subtle, but important. The first induction, from a belief of 0% to 9% is impossible to obtain from a single probability distribution P and violates the axioms of belief update. When Q_2 is impossible according to P, it remains impossible after observing evidence. Hence, the Bayesian learning paradigm fails in practice. More precisely, given a PDB at time t, such systems gather data D^t to obtain a new model $\mathrm{P}^{t+1}(.) = \mathrm{P}^t(. \mid D^t)$. Systems continuously add facts f, that is, set $\mathrm{P}^{t+1}(f) > 0$, whereas previously $\mathrm{P}^t(f) = 0$; an impossible induction for Bayesian learning[1].

Differently, let us consider knowledge base completion, i.e., the task of predicting new facts based on the existing facts in the knowledge base. One approach to knowledge base completion is to learn a probabilistic model from training data. Consider for example a probabilistic rule [12, 13] of the form

$$\mathsf{Costars}(x, y) \overset{0.8}{\longleftarrow} \mathsf{StarredIn}(x, z), \mathsf{StarredIn}(y, z), \mathsf{Couple}(x, y).$$

encoding the fact that if the query $\mathsf{StarredIn}(x, z), \mathsf{StarredIn}(y, z), \mathsf{Couple}(x, y)$ succeeds on a database, there is an 80% probability that we should derive the fact $\mathsf{Costars}(x, y)$. To evaluate the quality of this rule to predict the Costars relation, the standard approach would be to take the current probabilistic database together with labeled training data:

$$\mathcal{D} = \{\mathsf{Costars}(\mathsf{will\_smith}, \mathsf{jada\_smith}), \mathsf{Costars}(\mathsf{pitt}, \mathsf{jolie})\},$$

and quantify the conditional likelihood of the rule [44]. However, the rule predicts the probability zero for the fact $\mathsf{Costars}(\mathsf{will\_smith}, \mathsf{jada\_smith})$, as it is missing from the database. The rule gets the worst likelihood score of zero, regardless of its performance on other facts in the training data. Indeed, the semantics tells us that the absence of a single tuple can make $\mathsf{Costars}(\mathsf{will\_smith}, \mathsf{jada\_smith})$ impossible, invalidating the entire rule, which is otherwise highly accurate.

Another high-level task is to mine frequent patterns in the knowledge base. Given the probabilistic database the goal would, for instance, be to find interesting patterns, such as the pattern that many couples star in the same movie, and report it to the data miner. Again, the CWA will underestimate the expected frequencies of these patterns, and stand in the way of progress [45].

[1] Our goal is to highlight the consequences of the CWA in the Bayesian learning paradigm. We acknowledge, however, that the problems related to fact acquisition in knowledge bases are deeper. One may argue, for instance, that fact acquisition should be interpreted as another type of belief revision task, which is not necessarily Bayesian. While such discussions are very important, they are beyond the focus of the current paper, and require an independent, and a dedicated study.

Finally, we note that most of the automatically constructed PDBs are hardly probabilistic in the sense that most facts have a very high probability (highly skewed towards one), placing PDBs into an almost crisp setting. The underlying reason is that these systems retain only a small fraction of the discovered facts. Facts with a probability below a threshold are discarded. This mode of operation is not an oversight, but in most cases, a necessity. It is not always possible to retain all facts. Consider, for instance, the Sibling relation over a domain of 7 billion people. Storing a single-precision probability for all Sibling facts would require 196 exabytes of memory; two orders of magnitude more than the estimated capacity available to Google [46]. Moreover, the distribution of probabilities for such a closed-world database would be vastly different from the current ones, i.e., highly skewed towards zero. This issue of truncating and polynomial blow-up is inherent to probabilistic knowledge bases and needs to be acknowledged in their semantics.

4. Open-World Probabilistic Databases

We observe that large-scale knowledge bases are *incomplete* by their nature and systems used to build such knowledge bases should incorporate these characteristics into the query semantics. A feasible approach is to relax the probabilities of facts that are not in the database to a default probability interval, which is very different from the closed-world assumption of PDBs that requires the probabilities of such facts to be zero. Our proposal on open-world probabilistic databases builds on the theory of imprecise probabilities to allow such default, interval-based probabilities for the atoms that are not in the database. Syntactically, an open-world probabilistic database is a pair of a probabilistic database and a predefined threshold value.

Definition 4.1 (syntax)**.** An *open-world probabilistic database (OpenPDB)* is a pair $\mathcal{G} = (\mathcal{P}, \lambda)$, where $\mathcal{P}$ is a probabilistic database and λ is any rational number in $[0, 1]$.

The semantics of OpenPDBs is based on completing probabilistic databases. Intuitively, an OpenPDB denotes a partial specification over a vocabulary and needs to be completed by assigning a probability value from an interval $[0, \lambda]$ to each of the open atoms.

Definition 4.2 (completion)**.** A *λ-completion* of a probabilistic database $\mathcal{P}$ is another probabilistic database that is obtained as follows. For each atom t that does *not* appear in $\mathcal{P}$, we add an atom $\langle t : p\rangle$ to $\mathcal{P}$ for some $p \in [0, \lambda]$.

An OpenPDB induces a set of PDBs, each of which differs in the probabilities of the open atoms. Therefore, while a closed probabilistic database induces a unique probability distribution, an OpenPDB induces a (credal) set of probability distributions. A *credal set* is a closed convex set of probability distributions over a shared set of random variables.

Definition 4.3 (OpenPDBs)**.** An open probabilistic database $\mathcal{G} = (\mathcal{P}, \lambda)$ induces a credal set of probability distributions $\mathrm{K}_{\mathcal{G}}$ such that distribution P belongs to $\mathrm{K}_{\mathcal{G}}$ if and only if P is induced by some λ-completion of the PDB $\mathcal{P}$.

Intuitively, an OpenPDB represents all possible ways to extend a PDB with new atoms from the open world, with the restriction that the probability of these unknown atoms can never be larger than λ. When it is clear from the context, we will say completion instead of λ-completion.

Example 4.4. Consider again the PDB $\mathcal{P}_m$ from our running example. The pair $\mathcal{G}_c = (\mathcal{P}_m, 0.5)$ denotes an OpenPDB where open tuples can have the probability *at most* 0.5, shown in Table 3. Clearly, there are infinitely many possible completions of $\mathcal{G}_c$. Consider, for instance, the following completions:

$$\mathcal{P}_0 = \mathcal{P}_m \cup \{\langle t : 0\rangle \mid t \textit{ is an open atom in the PDB } \mathcal{P}_m\},$$
$$\mathcal{P}_{0.5} = \mathcal{P}_m \cup \{\langle t : 0.5\rangle \mid t \textit{ is an open atom in the PDB } \mathcal{P}_m\}.$$

These completions are special since they uniformly set all of the open atoms to the *same probability* value. These two completions induce different probability distributions, both of which belong to $\mathrm{K}_{\mathcal{G}_c}$.

Query semantics has to also take into account sets of probability distributions, and provide query probabilities in terms of *upper and lower probability values*.

Table 3: The OpenPDB $\mathcal{G}_c = (\mathcal{P}_m, 0.5)$ induces an infinite set of PDBs. Rows depicted in orange color represent open atoms that can take on any rational probability value from the default probability interval $[0, 0.5]$.

StarredIn		P
will_smith	ali	0.9
will_smith	sharktale	0.8
jada_smith	ali	0.6
arquette	scream	0.7
pitt	mr_ms_smith	0.5
jolie	mr_ms_smith	0.7
jolie	sharktale	0.9
pitt	ali	[0, 0.5]
pitt	fightclub	[0, 0.5]
arquette	fightclub	[0, 0.5]
...	...	[0, 0.5]

Couple		P
arquette	cox	0.6
pitt	jolie	0.8
thornton	jolie	0.6
pitt	aniston	0.9
kunis	kutcher	0.7
will_smith	jada_smith	[0, 0.5]
arquette	jolie	[0, 0.5]
pitt	kutcher	[0, 0.5]
...	...	[0, 0.5]

Definition 4.5 (query semantics)**.** Let Q be a Boolean query and $\mathcal{G}$ be an OpenPDB. The *probability interval of* Q in the OpenPDB $\mathcal{G}$ is defined as $\mathrm{K}_{\mathcal{G}}(Q) = [\underline{\mathrm{P}}_{\mathcal{G}}(Q), \overline{\mathrm{P}}_{\mathcal{G}}(Q)]$, where

$$\underline{\mathrm{P}}_{\mathcal{G}}(Q) = \min_{\mathrm{P} \in \mathrm{K}_{\mathcal{G}}} \mathrm{P}(Q) \qquad \text{and} \qquad \overline{\mathrm{P}}_{\mathcal{G}}(Q) = \max_{\mathrm{P} \in \mathrm{K}_{\mathcal{G}}} \mathrm{P}(Q).$$

Let us illustrate these concepts on an example.

Example 4.6. Consider the OpenPDB $\mathcal{G}_c = (\mathcal{P}_m, 0.5)$ and an open atom t. We have $\underline{\mathrm{P}}_{\mathcal{G}_c}(t) = 0$ and $\overline{\mathrm{P}}_{\mathcal{G}_c}(t) = \lambda$ by the query semantics. For instance, the ground query $Q = \mathsf{StarredIn}(\mathsf{pitt}, \mathsf{fightclub})$ evaluates to probability zero in $\mathcal{P}_m$. As for $\mathcal{G}_c$, it is easy to see that $\underline{\mathrm{P}}_{\mathcal{G}_c}(Q) = 0$ and $\overline{\mathrm{P}}_{\mathcal{G}_c}(Q) = 0.5$. The lower probability of Q remains the same due to the completion that assigns all open atoms the probability 0, while the upper probability increases due to the completion $\mathcal{P}_{0.5}$, shown earlier.

Our approach is analogous to the open world assumption defined over classical databases [16], where a database no longer corresponds to a single interpretation, but rather to the set of interpretations that extend it. A similar effect is achieved by OpenPDBs: an open probabilistic database no longer corresponds to a single distribution, but to the set of distributions that extend it. In restricting the probabilities of open atoms to lie in $[0, \lambda]$, OpenPDBs follow a rich literature on interval-based probabilities [47], which is also employed in credal networks [48]. Note also that setting a default probability interval is a form of default reasoning [49].

4.1. OpenPDBs in Practice

We discuss the implications of the open-world semantics, and compare it to closed-world PDBs. In particular, we revisit the motivating examples provided in Section 3, and highlight the differences in OpenPDBs.

4.1.1. Distinguishing Queries in OpenPDBs

In Section 3, we argued that the closed-world semantics fails to distinguish a class of queries which intuitively differ. Let us evaluate these queries in OpenPDBs, starting with the problem of distinguishing different levels of specificity.

Example 4.7 (specificity)**.** Consider again the following queries:

$$Q_1(x, y) = \exists z\, \mathsf{StarredIn}(x, z) \wedge \mathsf{StarredIn}(y, z) \wedge \mathsf{Couple}(x, y),$$
$$Q_2 = \mathsf{StarredIn}(x, z), \mathsf{StarredIn}(y, z), \mathsf{Couple}(x, y).$$

In Example 3.5, we have noted that both the query $Q_1(\mathsf{pitt}, \mathsf{jolie})$ and Q_2 evaluate to the same probability in the PDB $\mathcal{P}_m$. Since $Q_1(\mathsf{pitt}, \mathsf{jolie}) \models Q_2$, we argued that it was more reasonable to expect that $\mathrm{P}(Q_2) > \mathrm{P}(Q_1(\mathsf{pitt}, \mathsf{jolie}))$,

assuming our knowledge is not complete. This is indeed the case for the OpenPDB $\mathcal{G}_c = (\mathcal{P}_m, 0.5)$ for upper probabilities: $\overline{\mathrm{P}}_{\mathcal{G}_c}(Q_2) > \overline{\mathrm{P}}_{\mathcal{G}_c}(Q_1(\mathsf{pitt}, \mathsf{jolie}))$ since there are many worlds with non-zero probability that entail Q_2 but not $Q_1(\mathsf{pitt}, \mathsf{jolie})$. Notice that the lower probabilities remain unchanged.

It is hence possible to distinguish a *query* from a particular *instance of this query*, by comparing their respective upper probabilities in OpenPDBs. A similar argument applies to queries with varying level of support.

Example 4.8 (support)**.** Recall that $Q_1(\mathsf{will\_smith}, \mathsf{jada\_smith})$ has more support than $\overline{\mathrm{P}}(Q_1(\mathsf{thornton}, \mathsf{aniston}))$ in $\mathcal{P}_m$, as less additional facts are needed in $\mathcal{P}_m$ to satisfy $Q_1(\mathsf{will\_smith}, \mathsf{jada\_smith})$. Both queries evaluate to probability zero in $\mathcal{P}_m$, as identified in Example 3.6. Let us evaluate the queries in the OpenPDB $\mathcal{G}_c = (\mathcal{P}_m, 0.5)$, where the upper probability of open facts is bounded by 0.5, and hence lower than the probability of existing facts in $\mathcal{P}_m$. Then,

$$\overline{\mathrm{P}}(Q_1(\mathsf{will\_smith}, \mathsf{jada\_smith})) > \overline{\mathrm{P}}(Q_1(\mathsf{thornton}, \mathsf{aniston})) > 0,$$

as $\mathcal{P}_m$ requires fewer additional facts in a completion to satisfy $Q_1(\mathsf{will\_smith}, \mathsf{jada\_smith})$.

Thus, it is possible to distinguish queries that do not have a matching answer, as such queries typically have varying levels of support in the database. We also observed that an unsatisfiable query is, in some cases, as likely as a satisfiable one in the closed world. How are such queries evaluated in the open world?

Example 4.9 (satisfiability)**.** Recall from Example 3.7 the query $Q_1(\mathsf{will\_smith}, \mathsf{jada\_smith})$ as well as the query $\mathsf{StarredIn}(x, y) \wedge \neg\mathsf{StarredIn}(x, y)$ evaluates to probability zero on the PDB $\mathcal{P}_m$. In the open-world setting, the upper probability of a satisfiable query is always greater than the upper probability of an unsatisfiable query. Clearly, any unsatisfiable query still has a zero upper probability, because it is false in all completions.

These synthetic examples underline the difference in the semantics of PDBs and OpenPDBs. Do we really encounter similar examples in real-world data, which can benefit from an open-world perspective? To elaborate more on this question, we have extracted a portion from the NELL database concerning movies, actors, directors, etc. We conclude this subsection with this example.

Example 4.10 (real-world data)**.** Consider the following queries constructed based on a portion of the NELL database:

$$\begin{aligned}
Q_1 &= \mathsf{Actor}(\mathsf{pattinson}) \wedge \mathsf{Workedfor}(\mathsf{pattinson}, \mathsf{hardwicke}) \wedge \mathsf{Director}(\mathsf{hardwicke}),\\
Q_2 &= \exists x\, \mathsf{Actor}(x) \wedge \mathsf{StarredIn}(x, \mathsf{trainspotting}) \wedge \mathsf{Movie}(\mathsf{trainspotting}) \wedge \neg\mathsf{Director}(x),\\
Q_3 &= \exists x\, \mathsf{Actor}(\mathsf{pattinson}) \wedge \mathsf{Workedfor}(\mathsf{pattinson}, x) \wedge \mathsf{Director}(x).
\end{aligned}$$

All of the above queries have probability zero on the NELL database, yet we know they correspond to factually true statements. These queries, however, can be distinguished in an open-world setting, as they have varying levels of support. For example, we observe that Q_1 entails Q_3, and posing these queries in the open-world setting, we indeed obtain $\overline{\mathrm{P}}(Q_3) > \overline{\mathrm{P}}(Q_1)$ for any non-zero threshold λ. For instance, $\overline{\mathrm{P}}(Q_3) = 0.82$ and $\overline{\mathrm{P}}(Q_1) = 0.51$ for $\lambda = 0.3$. The query Q_2 finds actors that starred in the movie Trainspotting and did not direct a movie. Interestingly, there is no world satisfying this query in the (closed-world) NELL database. Evaluating Q_2 in OpenPDBs yields $\overline{\mathrm{P}}(Q_2) = 0.98$ and $\overline{\mathrm{P}}(Q_2) = 0.78$ with thresholds 0.7 and 0.3, respectively. These answers are clearly more in line with what one would expect.

4.1.2. Learning, Mining, and Knowledge Base Completion in OpenPDBs

We argued that the Bayesian learning paradigm is not in line with the CWA: as all open atoms are assigned the probability zero by the CWA, the principles of Bayesian learning are continuously violated while extending the knowledge base with new facts. The open-world semantics avoids this problem since (i) our initial belief consists of a set of probability distributions instead of a single one, and (ii) open atoms can take on probabilities from a default interval, i.e., not necessarily the probability zero.

We also argued that CWA permeates knowledge base completion, mining, and evaluation tasks, where we want to learn new facts to add to the database, using the facts that are already present. Recall the probabilistic rule [12, 13]:

$$\mathsf{Costars}(x, y) \overset{0.8}{\longleftarrow} \mathsf{StarredIn}(x, z), \mathsf{StarredIn}(y, z), \mathsf{Couple}(x, y),$$

stating that if the query $\mathsf{StarredIn}(x, z), \mathsf{StarredIn}(y, z), \mathsf{Couple}(x, y)$ succeeds on a database, there is an 80% probability for deriving $\mathsf{Costars}(x, y)$. Unlike PDBs, we do not get a prediction score *zero* for $\mathsf{Costars}(\mathsf{will\_smith}, \mathsf{jada\_smith})$ in OpenPDBs, as there are completions that contain this fact. Thus, the rule does not get the worst likelihood score of zero, but a higher likelihood, based on any completion that contains the fact $\mathsf{Costars}(\mathsf{will\_smith}, \mathsf{jada\_smith})$.

Finally, while motivating the need for open-world probabilistic databases, we argued that the issue of truncating and polynomial blow-up is inherent to probabilistic knowledge bases, and hence needs to be acknowledged in their semantics, since it is not always possible to retain all facts in the database whose probability fall below a certain threshold. The λ-value in OpenPDBs precisely represents this threshold.

5. Probabilistic Query Evaluation

Computing the probability of a query is a computationally demanding task in PDBs. We recall existing results for decision problems in PDBs and then present our results on OpenPDBs.

5.1. Query Evaluation in Probabilistic Databases

Dalvi and Suciu proved a dichotomy result for probabilistic query evaluation that classifies UCQs as either being computable in polynomial time, or #P-hard [50]. We are interested in the decision problem of probabilistic query evaluation, as defined next.

Definition 5.1 (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 $\mathrm{P}_{\mathcal{P}}(Q) > p$. We write $\mathrm{PQE}(Q)$ to denote probabilistic query evaluation for a fixed query Q. PQE can be defined over a particular query language instead of a specific query, in which case, we write $\mathrm{PQE}(\mathcal{L})$ to define PQE on the class $\mathcal{L}$ of queries.

Whenever the probabilistic database is clear from the context, we simply write $\mathrm{P}(Q)$ in place of $\mathrm{P}_{\mathcal{P}}(Q)$.

Remark 1. It is important to note that the decision problems $\mathrm{PQE}(Q)$ and $\mathrm{PQE}(\mathcal{L})$ are different. The former is more specific than the latter. Specifically, we can define the classes of instances captured by the respective decision problems as follows:

$$\mathrm{PQE}(Q) = \{\textit{Decide whether } \mathrm{P}_{\mathcal{P}}(Q) > p? \mid \mathcal{P} \textit{ is a PDB, } p \in [0, 1)\},$$
$$\mathrm{PQE}(\mathcal{L}) = \{\textit{Decide whether } \mathrm{P}_{\mathcal{P}}(Q) > p? \mid \mathcal{P} \textit{ is a PDB, } p \in [0, 1), Q \in \mathcal{L}\}.$$

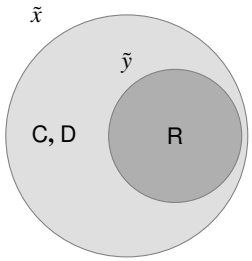
That is, $\mathrm{PQE}(\mathcal{L}) = \bigcup_{Q \in \mathcal{L}} \mathrm{PQE}(Q)$.

Remark 2. For our complexity analysis, we follow standard notions of data, and (bounded-arity) combined complexity, as explained in Section 2: data complexity of probabilistic query evaluation is calculated only based on the size of the probabilistic database, i.e., the query is fixed. The combined complexity of query evaluation is calculated by considering all the components, i.e., the probabilistic database, and the query are part of the input. For (bounded-arity) combined complexity, we additionally assume that the maximum arity of the predicates is bounded by an integer constant[2]. Within the scope of this paper, we always assume that the threshold value $p \in [0, 1)$, and the probability values from the interval $[0, 1]$ are always given as *rational* numbers. We also allow the threshold value p to depend on the input. This is a reasonable assumption since the probability computation necessarily depends on the data.

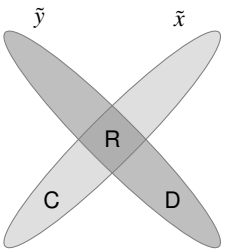
We defined the decision problems based on the comparison operator $>$, but since we focus on probabilistic databases, where the probabilities are always directly encoded in the input atoms, it is always possible to reduce the test for $\geq$ to the test for $>$, and vice versa.

Lemma 5.2. *For any PDB $\mathcal{P}$, and query Q, there exists a value ϵ which is polynomial in the size of $\mathcal{P}$ such that deciding $\mathrm{P}(Q) > p$ can be reduced to deciding $\mathrm{P}(Q) \geq p + \epsilon$, and deciding $\mathrm{P}(Q) \geq p$ can be reduced to deciding $\mathrm{P}(Q) \geq p - \epsilon$ in polynomial time. Similarly, deciding $\mathrm{P}(Q) < p$ can be reduced to deciding $\mathrm{P}(Q) \leq p - \epsilon$, and deciding $\mathrm{P}(Q) \leq p$ can be reduced to deciding $\mathrm{P}(Q) < p + \epsilon$ in polynomial time.*

[2]Clearly, $\mathrm{PQE}(Q)$ only applies to data complexity, since the query is fixed by definition. $\mathrm{PQE}(\mathcal{L})$ applies to all measures considered, e.g., in data complexity, we are free to choose any query $Q \in \mathcal{L}$ for $\mathrm{PQE}(\mathcal{L})$, but the query must still be fixed.



(a) $Q_{\mathrm{H}} = \exists x, y\, \mathsf{C}(x) \wedge \mathsf{R}(x, y) \wedge \mathsf{D}(x)$



(b) $Q_{\mathrm{NH}} = \exists x, y\, \mathsf{C}(x) \wedge \mathsf{R}(x, y) \wedge \mathsf{D}(y)$

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

We state Lemma 5.2 for PDBs, but it is easy to generalise this to many other related models, including OpenPDBs. This lemma gives us some liberty in the use of the operators $\geq$ and $>$. Hence, in some proofs, we will use, e.g., $\geq$ and $>$ interchangeably.

The data complexity of query evaluation depends heavily on the structure of the query. It is common to say that a query is *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 original result of Dalvi and Suciu.

Corollary 5.3 ([50])**.** *Let Q be a* UCQ*. Then,* PQE(Q) *is either in* P *or it is* PP*-complete for PDBs in data complexity under polynomial-time Turing reductions.*

Just like the original dichotomy, this result holds under polynomial-time Turing reductions. We use the same terminology also for the decision problem: we say that a query Q is *safe* if PQE(Q) is in P, and *unsafe*, otherwise.

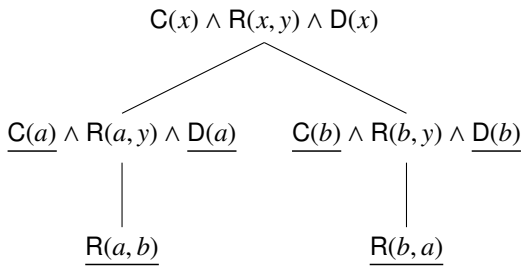
Dalvi and Suciu [51] proved the *small dichotomy result*, which applies to a subclass of conjunctive queries. As it gives nice insights on the larger dichotomy result [50], and allows us to introduce the basic notions relevant to this paper, we present this result in more detail. The small dichotomy 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*, and it is safe, otherwise. Therefore, it is crucial to understand *hierarchical* and *nonhierarchical* queries.

Definition 5.4 (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*.

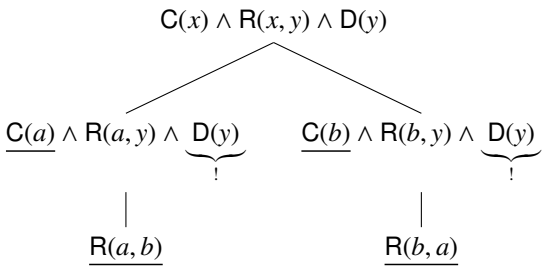
Let us consider the query $Q_{\mathrm{NH}} = \exists x, y\, \mathsf{C}(x) \wedge \mathsf{R}(x, y) \wedge \mathsf{D}(y)$. It is easy to see that this query is not hierarchical, since (i) the relation R occurs in both covers $\tilde{x}$ and $\tilde{y}$ (as depicted in Figure 1b), and neither of the covers is a subset of one another. This simple join query is already unsafe [51]. 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 $Q_{\mathrm{H}} = \exists x, y\, \mathsf{C}(x) \wedge \mathsf{R}(x, y) \wedge \mathsf{D}(x)$, as shown in Figure 1a, is yet another example of a safe query.

The intuition behind a safe query is the query being recursively decomposable into sub-queries such that each sub-query is probabilistically independent. For example, the query Q_{H} admits a decomposition: 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 the observation for the independence is also valid for all different groundings of Q_{H}. For example, the groundings $Q_{\mathrm{H}}[x/a]$ and $Q_{\mathrm{H}}[x/b]$, 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 2a in terms of a tree. The key ingredient



(a) Different branches of the tree do not share an atom, which ensures independence.



(b) Different branches of the tree share D-atoms, which makes them dependent.

Figure 2: Decomposition trees of a safe (a) and an unsafe query (b), respectively. In each of the trees, left branch corresponds to the grounding $[x/a, y/b]$, and right branch corresponds to the grounding $[x/b, y/a]$.

in this example is related to the variable x, which serves as a separator variable and allows us to further simplify the query.

Definition 5.5 (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.

For example, 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. That is, two different groundings $Q_{\text{NH}}[x/a]$ and $Q_{\text{NH}}[x/b]$ 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 2b. The small dichotomy theorem uses other rules of probability theory to further simplify the query as we illustrate next.

Example 5.6. Consider the hierarchical query $Q_{\text{H}} = \exists x, y\, \mathsf{C}(x) \wedge \mathsf{R}(x, y) \wedge \mathsf{D}(x)$. To compute the probability of Q_{H}, we first apply the decomposition based on the separator variable x, which yields

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

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

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

The dichotomy for UCQs is much more intricate and a characterization of safe queries is unfortunately not easy. Thus, an algorithm is given to compute the probability of all safe queries by recursively applying the simplification rules on the query [50]. This algorithm is complete, i.e., when the algorithm fails on the query, then the query is unsafe. Later, a version of the algorithm that targets ∀CNF queries, called LIFT$^{\text{R}}$, was proposed [52], which is also complete for that query class.

5.2. Query Evaluation in Open-World Probabilistic Databases

We now study probabilistic query evaluation in OpenPDBs, focusing in the following decision problems that extend probabilistic query evaluation to consider lower and upper probabilities for queries.

Definition 5.7 (upper, lower probabilistic query evaluation)**.** Given an OpenPDB $\mathcal{G}$, a query Q and a value $p \in [0, 1)$, *upper probabilistic query evaluation*, denoted $\overline{\text{PQE}}$, is to decide whether $\overline{\text{P}}_{\mathcal{G}}(Q) > p$ and *lower probabilistic query evaluation* denoted $\underline{\text{PQE}}$, is to decide whether $\underline{\text{P}}_{\mathcal{G}}(Q) > q$. We write $\overline{\text{PQE}}(Q)$ (resp., $\underline{\text{PQE}}(Q)$) to denote upper (resp., lower) probabilistic query evaluation for a fixed query Q. Both $\overline{\text{PQE}}$ and $\underline{\text{PQE}}$ can be defined over a particular query language; thus, we write $\overline{\text{PQE}}(\mathcal{L})$ (resp., $\underline{\text{PQE}}(\mathcal{L})$) to define $\overline{\text{PQE}}$ (respectively, $\underline{\text{PQE}}$) on the class of $\mathcal{L}$ queries.

OpenPDBs model an infinite set of PDBs, and it may seem like an unsurmountable task to efficiently compute the probability intervals $\text{K}_{\mathcal{G}}(Q)$. As we show later, the problem can be simplified to consider only *extremal* probability distributions that are obtained by setting the probability values of all elementary events to one of the extreme points.

Definition 5.8. Let $\mathcal{G} = (\mathcal{P}, \lambda)$ be an arbitrary OpenPDB; we call a probability distribution $\text{P} \in \text{K}_{\mathcal{G}}$ an *extremal distribution* if for all open atoms t, either $\text{P}(t) = \lambda$ or $\text{P}(t) = 0$ holds.

We now show that to compute the upper and lower probability bounds, it is sufficient to consider the distributions, where open atoms can take on the probability λ or 0 , i.e., no intermediate choices need to be examined.

Theorem 5.9. *Let $\mathcal{G}$ be an arbitrary OpenPDB and Q an* FO *query. There exist extremal distributions $\underline{\text{P}}, \overline{\text{P}} \in \text{K}_{\mathcal{G}}$ such that $\underline{\text{P}}(Q) = \underline{\text{P}}_{\mathcal{G}}(Q)$, and $\overline{\text{P}}(Q) = \overline{\text{P}}_{\mathcal{G}}(Q)$.*

Clearly, there are exponentially many extremal distributions, each of which sets the probability of (at least) one atom to a different extreme. Thus, Theorem 5.9 suggests a naïve query answering algorithm: generate all extreme distributions P, compute $\text{P}(Q)$, and report the minimum and maximum. This is very inefficient, as it requires exponentially many calls in the number of open-world atoms. Note that the monotonicity of unions of conjunctive queries allows us to further simplify query evaluation. In essence, we can simply choose the minimal (resp., maximal) bound for every atom and the resulting probability for the UCQ is ensured to be minimal (resp., maximal). Thus, the lower probabilities of conjunctive queries in OpenPDBs can be computed using a standard PDB algorithm. To compute the upper bounds, we can construct a new PDB from the OpenPDB, by adding all the open facts with default upper probabilities λ and simply reuse standard algorithms developed for PDBs.

Theorem 5.10. *Let $\mathcal{G} = (\mathcal{P}, \lambda)$ be an arbitrary OpenPDB, Q a* UCQ *and $\mathcal{P}_\lambda$ the completion that sets the probabilities of all open tuples to λ. Then, it holds that $\text{K}_{\mathcal{G}}(Q) = [\text{P}_{\mathcal{P}}(Q), \text{P}_{\mathcal{P}_\lambda}(Q)]$.*

Observe that in OpenPDBs, we can easily recover the upper (resp., lower) probability of a query from the lower (resp., upper) probability of its negation, as shown next.

Lemma 5.11. *Let $\mathcal{G} = (\mathcal{P}, \lambda)$ be an OpenPDB and Q a first-order query. It holds that $\overline{\text{P}}_{\mathcal{G}}(Q) = 1 - \underline{\text{P}}_{\mathcal{G}}(\neg Q)$ and $\underline{\text{P}}_{\mathcal{G}}(Q) = 1 - \overline{\text{P}}_{\mathcal{G}}(\neg Q)$.*

The construction given in Theorem 5.10 is still not very efficient, as it adds all the atoms to the PDB, which grows polynomially in the domain size. Unfortunately, this is impractical for PDBs with a large domain. Indeed, on the Sibling example from Section 3.1.2, the upper bound would have to be computed on a 196 exabyte closed-world PDB. Thus, an important question is whether this grounding can be avoided, as we investigate next.

Note that the lower probability of a UCQ in OpenPDBs can be computed with any closed-world PDB algorithm. We present an algorithm, called $\textsc{Lift}^{\text{R}}_{\text{O}}$ (Algorithm 1), which can be used to compute the upper probability of a UCQ in OpenPDBs. $\textsc{Lift}^{\text{R}}_{\text{O}}$ performs operations on ∀CNF formulas, i.e., unions of CNF formulas. More specifically, $\textsc{Lift}^{\text{R}}_{\text{O}}$ takes as input an OpenPDB $\mathcal{G} = (\mathcal{P}, \lambda)$ over a domain, and a negated UCQ Q as an input, and outputs $\underline{\text{P}}_{\mathcal{G}}(Q)$. Since we can compute the upper probability of any UCQ, from the lower probability of its negation, i.e., $\overline{\text{P}}_{\mathcal{G}}(Q) = 1 - \underline{\text{P}}_{\mathcal{G}}(\neg Q')$ for any query Q by Lemma 5.11, this algorithm can be used directly to compute the upper probability of a given UCQ in OpenPDBs.

Preprocessing. The algorithm assumes that any input query is preprocessed such that (i) it does not contain any constant symbols and (ii) all variables appear in the same order in each predicate occurrence in Q. This preprocessing can be done in polynomial time in data complexity and therefore it is efficient. Preprocessing is necessary for several reasons; most importantly, in order to capture all safe queries by an algorithm. Let us first present the details of the preprocessing.

Definition 5.12 (shattering, ranking)**.** A first-order query Q is *shattered* if it does not contain any constants. A first-order query Q is *ranked* if there exists a total order $\prec$ on its variables such that for every atom $\mathsf{R}(\vec{x})$ of arity $k \geq 2$, whenever x_i, x_j occur in $\mathsf{R}(\vec{x})$ and x_i occurs before x_j then $x_i \prec x_j$; in particular, no atom contains the same variable twice.

Let us briefly illustrate the process of ranking on a simple example.

Example 5.13. Consider the query $\exists x, y\ \mathsf{S}(x, y) \wedge \mathsf{S}(y, x)$, which is not ranked, since the variables x and y occur in different orders in the same predicate. To rank this query, we first split the predicate S into three predicates $\mathsf{S}_{x<y}$, $\mathsf{S}_{y<x}$ and $\mathsf{S}_{x=x}$. We then define a total order ρ on the database constants (say, a and b) and split the S-atoms in the PDB such that all occurrences of

- $\mathsf{S}(a, b)$ is replaced with $\mathsf{S}_{x<y}(a, b)$ if $a \prec b$,
- $\mathsf{S}(a, b)$ is replaced with $\mathsf{S}_{y<x}(b, a)$ if $b \prec a$,
- $\mathsf{S}(a, a)$ is replaced with $\mathsf{S}_{x=x}(a)$,
- $\mathsf{S}(b, b)$ is replaced with $\mathsf{S}_{x=x}(b)$.

This ensures that all appearances of the variables in some atom respect the order. Then, the ranking of the example query $\exists x, y\ \mathsf{S}(x, y) \wedge \mathsf{S}(y, x)$ is given as: $\exists x, y\ (\mathsf{S}_{x<y}(x, y) \wedge \mathsf{S}_{y<x}(x, y)) \vee \exists x \mathsf{S}_{x=x}(x)$. Intuitively, this preprocessing partitions the predicates and the corresponding atoms in the database with respect to some ordering. It is easy to see that this transformation preserves the semantics; for details, we refer to the dichotomy result of Dalvi and Suciu [50].

It has been shown that the preprocessing does not affect the probability computation in PDBs: let Q be a query, $\mathcal{P}$ be a PDB, and Q^r, $\mathcal{P}^r$, their rankings. Then, it holds that $\mathrm{P}_{\mathcal{P}}(Q) = \mathrm{P}_{\mathcal{P}^r}(Q^r)$. This clearly translates to OpenPDBs, since once a λ-completion is chosen for all open atoms, we obtain a single ranked PDB (assuming we added a polynomial number of atoms to the database with zero probability). Thus, it is easy to conclude that this preprocessing preserves the semantics also for OpenPDBs.

Ranking can be done in linear time in PDBs, but for OpenPDBs, this is unfortunately not always the case, since we also have to consider the open atoms. Thus, in the worst case, ranking will cause a polynomial blow-up seems to be *unavoidable* in OpenPDBs. Hence, it remains *open* whether the overall polynomial cost can be avoided in OpenPDBs. We note, however, that ranking is only needed for repeating relation symbols, i.e., if the query is self-join free, then this process is not needed. Therefore, this preprocessing can be limited to repeating relation symbols so as to avoid to polynomial blow-up as much as possible. In the presented algorithm, we assume that the query and the PDB are preprocessed in this way.

Lifted Inference Algorithm. Algorithm 1 is an adaptation of the $\textsc{Lift}^{\mathrm{R}}$ algorithm [52], which goes back to the algorithm givcen by Dalvi and Suciu [50]. This algorithm is called $\textsc{Lift}^{\mathrm{R}}_{\mathrm{O}}$, where O stands for open.

Step 0. Recall that the given UCQ is negated in the preprocessing to obtain a ∀CNF query Q. As a result, all atoms appear negatively in Q. The *base case* of the algorithm applies when the query is simply a negated ground atom $\neg t$. In this case the probability of the query is trivial to compute: if the atom appears in the PDB with a probability p, then the algorithm returns $(1 - p)$; otherwise, it is an open atom and the algorithm returns $(1 - \lambda)$.

Step 1. The *first step* is to rewrite the query Q into a union (disjunction) of CNF sentences, or UCNF. For example, consider the CNF formula: $(\mathsf{R}(x) \vee \mathsf{S}(y, z)) \wedge (\mathsf{S}(x, y) \vee \mathsf{T}(x))$ which can be rewritten as the disjunction of the CNF formulas:

$$\mathsf{R}(x) \wedge \mathsf{S}(x, y) \quad \text{union} \quad \mathsf{R}(x) \wedge \mathsf{T}(x) \quad \text{union} \quad \mathsf{S}(y, z) \wedge \mathsf{S}(x, y) \quad \text{union} \quad \mathsf{S}(y, z) \wedge \mathsf{T}(x).$$

The intuition behind this transformation is to produce multiple disjuncts from the given CNF in order to make (disjunctive) independencies explicit (if there are any). Note that such a rewriting does not always produce multiple disjuncts, in which case, the formula is clearly also a CNF.

Algorithm 1 $\mathsf{Lift}^{\mathsf{R}}_{\mathsf{O}}(Q, \mathcal{P}, \lambda, \Delta)$, abbreviated by $\mathbf{L}(Q, \mathcal{P})$

Require: A negated UCQ Q, an OpenPDB $\mathcal{G} = (\mathcal{P}, \lambda)$, and domain Δ.
Ensure: The lower probability $\underline{\mathrm{P}}(Q)$ in the OpenPDB $\mathcal{G} = (\mathcal{P}, \lambda)$ over domain Δ.

Step 0 *Base of recursion*
  if $Q = \neg t$, where t is a ground atom **then**
    if $\langle t : p \rangle \in \mathcal{P}$ **then return** $(1 - p)$ ▷ Closed atoms
    else return $(1 - \lambda)$ ▷ Open atoms
Step 1 *Rewriting of the query*
  Convert Q to UCNF: $Q_{\mathrm{UCNF}} = (\forall \vec{x}\, Q_1) \vee \ldots \vee (\forall \vec{y}\, Q_m)$
Step 2 *Decomposable disjunction* ▷ Probabilistically independent disjuncts
  if $m > 1$ and $Q_{\mathrm{UCNF}} = Q_1 \vee Q_2$ where $Q_1 \perp Q_2$ **then**
    $q_1 \leftarrow \mathbf{L}(Q_1, \mathcal{P}_{|_{Q_1}})$ and $q_2 \leftarrow \mathbf{L}(Q_2, \mathcal{P}_{|_{Q_2}})$
    return $1 - (1 - q_1) \cdot (1 - q_2)$
Step 3 *Inclusion-exclusion*
  Apply cancellations/minimizations on Q.
  if $m > 1$ but Q_{UCNF} has no independent sub-query Q_i **then**
    return $\sum_{s \neq \emptyset, s \subseteq [m]} (-1)^{|s|+1} \cdot \mathbf{L}(\bigwedge_{i \in s} Q_i, \mathcal{P}_{|_{\bigwedge_{i \in s} Q_i}})$ ▷ $[m] = \{1, \ldots, m\}$
Step 4 *Decomposable conjunction* ▷ Probabilistically independent conjuncts
  Convert Q back to CNF: $Q_{\mathrm{CNF}} = \forall \vec{x}\, Q_1 \wedge \ldots \wedge Q_k$
  if $Q = Q_1 \wedge Q_2$ where $Q_1 \perp Q_2$ **then**
    return $(\mathbf{L}(Q_1, \mathcal{P}_{|_{Q_1}}) \cdot \mathbf{L}(Q_2, \mathcal{P}_{|_{Q_2}}))$
Step 5 *Decomposable universal quantifier* ▷ Probabilistically independent projection
  if Q has a *separator variable* x **then**
    let E be all constants that appear as *x-argument* in $\mathcal{P}$
    $q_c \leftarrow \prod_{e \in E} \mathbf{L}(Q[x/e], \mathcal{P}_{|_{x=e}})$ ▷ Ground and recurse over known atoms
    $q_o \leftarrow \mathbf{L}(Q[x/e], \emptyset)$ for some $e \in \Delta \setminus E$ ▷ Recurse over a canonical open atom
    return $q_c \cdot q_o^{|\Delta \setminus E|}$ ▷ Generalize the computation to the size of the domain
Step 6 *Fail*

Step 2. The *second step* applies when the resulting UCNF has multiple disjuncts (or equivalently if it is not a CNF). The algorithm checks whether it is possible to partition the query into two UCNF formulas such that $Q = Q_1 \vee Q_2$, where Q_1 and Q_2 do not share any relational symbols, denoted $Q_1 \perp Q_2$, which ensures independence of Q_1 and Q_2. Then, it applies the probabilistic decomposition rule for disjunction:

$$\mathrm{P}(Q) = 1 - (1 - \mathrm{P}(Q_1)) \cdot (1 - \mathrm{P}(Q_2)).$$

It is easy to verify the correctness of this decomposition provided that Q_1 and Q_2 are independent terms which holds, as they do not share any relation symbols. The main idea in the second step (as well as in the remaining steps) is to recurse on simplified queries, using standard simplification rules of probability. Importantly, in the various recursions, the algorithm shrinks the set of atoms in the given PDB $\mathcal{P}$. Specifically, $\mathcal{P}_{|_Q}$ denotes the subset of $\mathcal{P}$, containing only atoms for the predicates that appear in Q.

Step 3. The *third step* also applies only when the UCNF has multiple disjuncts and recurses using the *inclusion-exclusion* principle:

$$\mathrm{P}(Q) = \sum_{s \neq \emptyset, s \subseteq [m]} (-1)^{|s|+1} \cdot \mathrm{P}(\wedge_{i \in s} Q_i).$$

The key aspect in this step is to apply cancellations before the inclusion-exclusion step. The idea is to remove redundancies from the query and minimize it by checking for CNF formulas that are implied by others. This can be

done using standard algorithms [53]. After simplifying the query, we also need to use the *Möbius function* to build an implication lattice, and then, count how many times to include each subquery using the Möbius function, and finally, remove subqueries whose inclusion-exclusion coefficients sum to 0 [50]. Importantly, note that these manipulations are only on the query and, therefore, independent of the database.

Step 4. In the *fourth step* the query is rewritten back as a CNF. Then, the algorithm checks for independent sets of clauses in the CNF such that $Q = Q_1 \wedge Q_2$, where Q_1 and Q_2 do not share any relational symbols. If this is the case, then it applies the probabilistic decomposition rule for conjunction:

$$\mathrm{P}(Q) = \mathrm{P}(Q_1) \cdot \mathrm{P}(Q_2).$$

Step 5. The *fifth step* is the workhorse of $\textsc{Lift}^{\mathrm{R}}_{\mathrm{O}}$, and the key difference with the $\mathsf{Lift}^{\mathsf{R}}$ algorithm [52]. It searches for a separator variable. The existence of a separator variable implies that for any two distinct instantiations e_1, e_2 of the separator, the queries $Q[x/e_1]$ and $Q[x/e_2]$ are independent. Hence, by multiplying $\overline{\mathrm{P}}(Q[x/e])$ for all e in the domain Δ, we obtain $\overline{\mathrm{P}}(Q)$.

The implementation of step five in $\textsc{Lift}^{\mathrm{R}}_{\mathrm{O}}$ performs one key optimization over this simple multiplication. First, note that x appears in exactly one argument position in Q for every predicate. We call these arguments the x-arguments. Step five partitions the constants in the domain into two sets: (i) the constants E that appear as x-arguments in the tuples in $\mathcal{P}$, and (ii) all other constants, denoted by $\Delta \setminus E$.

For (i), $\textsc{Lift}^{\mathrm{R}}_{\mathrm{O}}$ still enumerates all instantiations of x and computes their probability separately. For (ii), it suffices to compute the probability of a single instantiation of x. All instantiations with constants from $\Delta \setminus E$ will have the same probability, as they do not depend on the tuples in $\mathcal{P}$. The probability of their conjunction is computed by exponentiation. Moreover, in the recursive calls for $[x/e]$, we can pass along the subset of the atoms $\mathcal{P}_{|x=e}$ where all x-arguments are constant e.

Step 6. Finally, $\textsc{Lift}^{\mathrm{R}}_{\mathrm{O}}$ can fail in *step six*, yielding no answer, which implies that the query in unsafe, as we shall discuss in the next section.

6. Data Complexity Results

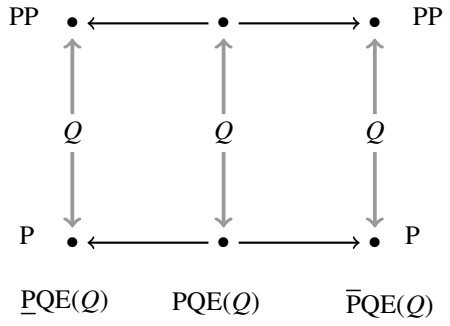
We first study data complexity, and obtain complexity results relative to different query languages under consideration. We start with an overview of the data complexity results.

6.1. Overview of the Data Complexity Results

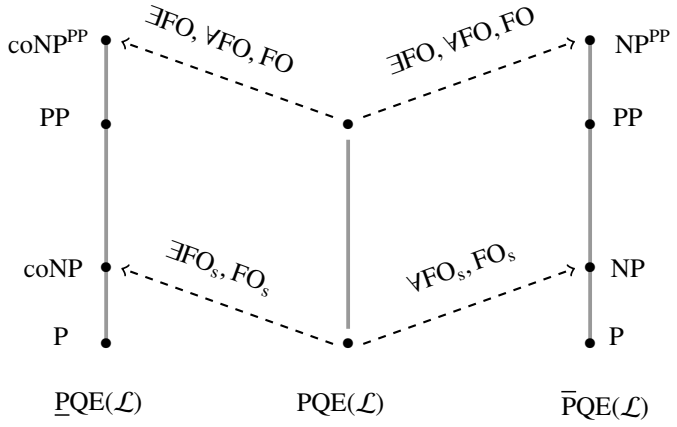
Our open-world semantics is supported by a *query evaluation algorithm* for UCQs. This class of queries, corresponding to monotone ∃DNF, is particularly well-behaved and the focal point of database research. Perhaps the largest appeal of PDBs comes from a dichotomy result by Dalvi and Suciu [50], perfectly delineating which unions of conjunctive queries can be answered efficiently in data complexity. Their algorithm runs in polynomial time for all efficient queries, called *safe queries*, and recognizes all others to be #P-hard (which translates into PP-hardness under polynomial-time Turing reductions).

We give an algorithm $\textsc{Lift}^{\mathrm{R}}_{\mathrm{O}}$, which extends the PDB algorithm of Dalvi and Suciu [50] and inherits its elegant properties: all safe queries run in polynomial time, and whenever our algorithm fails, then the query is PP-hard. Thus, for unions of conjunctive queries, we are able to show that the data complexity dichotomy in PDBs can be lifted to OpenPDBs, as depicted in Figure 3a.

Importantly, we show that $\textsc{Lift}^{\mathrm{R}}_{\mathrm{O}}$ runs in linear time in the size of the ranked OpenPDB (resp., PDB), under certain assumptions. More specifically, by the properties of $\textsc{Lift}^{\mathrm{R}}_{\mathrm{O}}$, we show that the original dichotomy of PDBs can be strengthened under mild assumptions, i.e., (i) unit arithmetic cost assumption, that is, the complexity of all arithmetic operations in the algorithmis fixed, and (ii) the domain of the input PDB is sorted, e.g., it is an integer domain. Hence, all safe queries can be computed in linear time in the size of the input PDBs. The fact that the original PDB dichotomy [50] can be strengthened in this way is perhaps not technically surprising. However, this practically significant observation has not been made earlier in the literature. It is open whether the linear-time computation can be extended to the case of OpenPDBs for the full class of safe UCQs. The algorithm $\textsc{Lift}^{\mathrm{R}}_{\mathrm{O}}$ runs in linear-time in



(a) Dichotomy for UCQs: Safe (resp., unsafe) UCQs coincide for all problems $\underline{\text{P}}\text{QE}(Q)$, $\text{PQE}(Q)$, and $\overline{\text{P}}\text{QE}(Q)$ for any $Q \in \text{UCQ}$.



(b) Data complexity results for the problems $\underline{\text{P}}\text{QE}(\mathcal{L})$, $\text{PQE}(\mathcal{L})$, and $\overline{\text{P}}\text{QE}(\mathcal{L})$ for the query classes $\mathcal{L} \in \{\exists\text{FO}, \forall\text{FO}, \text{FO}\} \cup \{\exists\text{FO}_\text{s}, \forall\text{FO}_\text{s}, \text{FO}_\text{s}\}$.

Figure 3: Data complexity map for OpenPDBs in comparison to PDBs. Figure 3a depicts the dichotomy results for UCQs: safe (resp., unsafe) UCQs for PDBs coincide with safe (resp., unsafe) UCQs for OpenPDBs. Figure 3b depicts the complexity results for more general query classes ∀FO, ∃FO, and FO, and their restricted safe fragments; for each such query class, the complexity of probabilistic query evaluation increases in OpenPDBs, compared to PDBs.

the size of the *preprocessed, ranked* OpenPDB, rather than the input OpenPDB, and this preprocessing can lead to a polynomial blow-up in the case of OpenPDBs. We note, however, that such preprocessing is not needed for, e.g. self-join-free queries, and, in this case, the linear-time algorithm applies also to OpenPDBs under the same assumptions.

We then extend our analysis to other query languages, which results in a richer complexity landscape. All results for ∃FO, ∀FO, and FO queries for upper and lower probabilistic query evaluation are depicted in Figure 3b (upper part). Our results suggest that the complexity of open-world reasoning can go up significantly with negation. Specifically, we first show that PQE(∃FO), PQE(∀FO), and PQE(FO) are all PP-complete in data complexity for PDBs. On the other hand, $\overline{\text{P}}\text{QE}(\exists\text{FO})$, $\overline{\text{P}}\text{QE}(\forall\text{FO})$, and $\overline{\text{P}}\text{QE}(\text{FO})$ are NP^{PP}-complete in data complexity for OpenPDBs. Similarly, the corresponding lower probabilistic query evaluation problems are shown to be coNP^{PP}-complete in data complexity for OpenPDBs.

Knowing that all safe (resp., unsafe) queries remain safe (resp., unsafe) in OpenPDBs for UCQs, we pose the following question: could this also be the case for more expressive queries, where the satisfaction relation is not monotone? To make this concrete, we denote by $\exists\text{FO}_\text{s}$, $\forall\text{FO}_\text{s}$, and FO_s, the subclasses of the query classes ∃FO, ∀FO, and FO, respectively, where each such subclass contains only queries, which are safe for PDBs. We note that the classification status of nonmonotone query classes still remain open in PDBs, i.e., there is no known classification for such general query classes. We show that the class of safe PDB queries are not preserved when we consider OpenPDBs. Specifically, we identify a $\forall\text{FO}_\text{s}$ query, which is safe for PDBs, but becomes NP-complete on OpenPDBs. All results for $\exists\text{FO}_\text{s}$, $\forall\text{FO}_\text{s}$, and FO_s queries for upper and lower probabilistic query evaluation are depicted in Figure 3b (lower part).

OpenPDBs are closely related to credal representations, and thus our complexity results align with that of credal networks which also show an increase from P to NP and from PP to NP^{PP} [54] compared to Bayesian networks [55]. Nevertheless, one source of hardness for probabilistic inference in credal networks is due to the conditional dependencies encoded in the network structure, which is very different from OpenPDBs, where the hardness stems from rich structure of queries.

6.2. Results for Unions of Conjunctive Queries

We start our analysis with UCQs, and discuss the implications of Algorithm 1, in detail. The original dichotomy [50] is supported by an algorithm similar to $\textsc{Lift}^{\text{R}}_{\text{O}}$: if this algorithm fails, then the query is #P-hard, and if it does not fail, it runs in polynomial time. This dichotomy-supporting algorithm has one major difference compared to $\textsc{Lift}^{\text{R}}_{\text{O}}$, aside from our support for open-world inference. When it applies the inclusion-exclusion step, it

performs cancellations to avoid computing some of the recursive steps. This is a key aspect of the algorithm that ensures efficiency for all safe queries. Based on Theorem 5.10 and Corollary 5.3 we can lift the dichotomy result for UCQ queries in PDBs to OpenPDBs.

Corollary 6.1 (dichotomy)**.** *Let Q be a* UCQ*. Then,* $\overline{\text{P}}\text{QE}(Q)$ *is either in* P *or it is* PP*-complete for OpenPDBs in data complexity under polynomial-time Turing reductions. Moreover, a* UCQ *Q is safe in OpenPDBs if and only if it is safe in PDBs.*

We already emphasized that the reduction given in Theorem 5.10 for the class of safe queries can be inefficient in practical terms as the construction results in a polynomial blow-up. Similarly, the preprocessing step in $\text{Lift}^{\text{R}}_{\text{O}}$ algorithm, can be polynomial. Ignoring the preprocessing, we show that, $\text{Lift}^{\text{R}}_{\text{O}}$ extended with cancellations in the inclusion-exclusion step, runs in linear time. For this to hold, we need two assumptions: (i) unit arithmetic cost assumption, that is, the complexity of all arithmetic operations in the algorithm is fixed, and (ii) the domain of the OpenPDB is sorted, e.g., it is an integer domain. Intuitively, the first assumption is needed since the cost of arithmetic operations can grow beyond linear in the size of the probability values. The second assumption is necessary to ensure that we always recurse over the subset of atoms, in an ordered way, so as to avoid revisiting them.

Theorem 6.2. *Let $\mathcal{G} = (\mathcal{G}, \lambda)$ an OpenPDB over an integer domain (or, any sorted domain). Assuming unit arithmetic cost, the following results hold. The algorithm* $\text{Lift}^{\text{R}}_{\text{O}}$ *runs in polynomial time in data complexity. The probability of any* safe *UCQ can be evaluated in* linear time *in the size of the* ranked *OpenPDB $\mathcal{G}$ in data complexity. Moreover, any safe UCQ which is self-join-free can be evaluated in* linear time *in the size of the input OpenPDB $\mathcal{G}$ in data complexity.*

The algorithm $\text{Lift}^{\text{R}}_{\text{O}}$ clearly runs in polynomial time in data complexity (while it answers Fail for unsafe queries). It is also easy to see that it is linear in the size of the *ranked* OpenPDB $\mathcal{G}$ in data complexity. Notice though, that the ranked OpenPDB can be polynomially larger than the original given one. The final statement states that, if the query is additionally self-join-free, in which case, ranking is not needed, then $\text{Lift}^{\text{R}}_{\text{O}}$ runs in *linear time* in the size of the input OpenPDB $\mathcal{G}$ in data complexity.

Theorem 6.2 implies that the algorithm Lift^{R}, which is a special case of $\text{Lift}^{\text{R}}_{\text{O}}$, also runs in linear time. While the preprocessing can be polynomial in OpenPDBs, it remains linear in PDBs. Overall, these imply a stronger dichotomy for PDBs.

Corollary 6.3. *Let Q be a* UCQ*, and $\mathcal{P}$ be a PDB over an integer domain (or, any sorted domain). Assuming* unit arithmetic cost*,* PQE(Q) *for PDBs over integer domains is either in linear time, or it is* PP*-complete.*

This result extends the original dichotomy for PDBs from polynomial time to linear time under the given assumptions. This observation appears to be novel in the PDB literature. The original algorithm [50] is not shown to be linear-time. Existing linear-time probabilistic query evaluation complexity results, see e.g. [56], do not apply to unions of conjunctive queries. The key insight behind our linear-time algorithm is the projection of the probabilistic database only on the relevant atoms for each recursive call. This concludes our data complexity analysis for unions of conjunctive queries.

6.3. Results Beyond Unions of Conjunctive Queries

We now focus on query languages that strictly contain UCQs, i.e., the query classes ∃FO, ∀FO, and FO. Let us start our data complexity analysis with PDBs. We give a general theorem which shows that the probabilistic query evaluation problem is PP-complete in PDBs for all query languages under consideration.

Theorem 6.4. PQE(∃FO)*,* PQE(∀FO)*, and* PQE(FO) *are* PP*-complete for PDBs in data complexity.*

For the membership results, the main idea is to code each world induced by a PDB into a nondeterministic Turing machine such that each world satisfying (resp., *not* satisfying) the query corresponds to a number of accepting (resp., rejecting) computation branches proportional to its probability. By additionally introducing artificial accept (resp., reject) branches, we ensure that the majority of the runs of the nondeterministic Turing machine answer yes if and only if the given instance of probabilistic query evaluation has a positive answer.

To show PP-hardness (under many one reductions) we give a reduction from counting the satisfying assignments of a Boolean formula: given a quantified Boolean formula of the form $\Phi \coloneqq \mathsf{C}^c\, x_1, \ldots, x_n\, \phi$, where C represents the

counting quantifier and $\phi = \phi_1 \wedge \cdots \wedge \phi_k$ is a propositional formula in 3CNF, defined over the variables $x_1, \ldots, x_n$, decide whether Φ is valid. Intuitively, this problem amounts to checking whether there are c assignments for $x_1, \ldots, x_n$ that satisfy ϕ, which is a PP-complete problem [33].

We proceed with the data complexity results for OpenPDBs. Let us start with a simple observation, which is very useful for OpenPDBs. By Lemma 5.11, we know that $\overline{\mathrm{P}}(Q) = 1 - \underline{\mathrm{P}}(\neg Q)$. In particular, this implies that:

$$\overline{\mathrm{P}}_{\mathcal{G}}(Q) > p \text{ if and only if it is } \textit{not} \text{ the case that } \underline{\mathrm{P}}_{\mathcal{G}}(\neg Q) \geq 1 - p,$$
$$\underline{\mathrm{P}}_{\mathcal{G}}(Q) > p \text{ if and only if it is } \textit{not} \text{ the case that } \overline{\mathrm{P}}_{\mathcal{G}}(\neg Q) \geq 1 - p,$$

for any OpenPDB $\mathcal{G}$, query Q and threshold value p. Recall that $\geq$ can be replaced with $>$ by Lemma 5.2. Then, since ∀FO and ∃FO queries are dual to each other, probabilistic query evaluation for these queries can be reduced to each other by taking the complement of the respective problem. In essence, all complexity results obtained for the problem $\overline{\mathrm{P}}$QE(∀FO) immediately hold for the complement of the problem $\underline{\mathrm{P}}$QE(∃FO), and vice versa. Similarly, all complexity results for the problem $\underline{\mathrm{P}}$QE(∀FO) hold for the complement of the problem $\overline{\mathrm{P}}$QE(∃FO), and vice versa. We refer to this as the *duality property* and use it to simplify the proofs of some of the theorems. Practically speaking, this allows us to state the results regarding both to ∃FO and ∀FO queries, while providing the proof details only for one of these classes. It is worthwhile to note that *lower* and *upper* probabilistic query evaluation for the same query language are *not* dual to each other, e.g., the result for $\overline{\mathrm{P}}$QE(∀FO) does not imply the result of $\underline{\mathrm{P}}$QE(∀FO), or vice versa, and so such results are proven separately, but they nevertheless rely on similar ideas.

We have shown that the data complexity dichotomy for UCQs can be lifted from PDBs to OpenPDBs: all safe (resp., unsafe) queries remain safe (resp., unsafe). Could this also be the case for more expressive queries, where the satisfaction relation is not monotone? There is no obvious way of determining the completion that maximizes (or, minimizes) the query probability for such query classes. Therefore, an intriguing question is, whether a nonmonotone query, which is safe in PDBs, can become hard for OpenPDBs?

We note that the classification status of nonmonotone query classes still remain open in PDBs, i.e., there is no known classification for such general query classes that perfectly delineates safe queries from unsafe ones; it is also open whether such a dichotomy exists at all. We are only interested in knowing whether the class of safe queries could possibly be preserved when we consider OpenPDBs. To make this concrete, let us denote by $\exists\mathrm{FO}_\mathrm{s}$, $\forall\mathrm{FO}_\mathrm{s}$, and FO_s, the subclasses of the query classes ∃FO, ∀FO, and FO, respectively, where each such subclass contains only safe queries from the general class. For instance, $\forall\mathrm{FO}_\mathrm{s} \subseteq \forall\mathrm{FO}$, and every query $Q \in \forall\mathrm{FO}_\mathrm{s}$ is safe in PDBs, i.e., its probability can be computed in polynomial time for any PDB. For these query classes, we obtain the following result.

Theorem 6.5. $\overline{\mathrm{P}}\mathrm{QE}(\mathrm{FO}_\mathrm{s})$ *is* NP*-complete and* $\underline{\mathrm{P}}\mathrm{QE}(\mathrm{FO}_\mathrm{s})$ *is* coNP*-complete for OpenPDBs in data complexity. Furthermore, there exists a query* $Q_{\mathrm{SAFE}} \in \forall\mathrm{FO}_\mathrm{s}$ *such that* $\overline{\mathrm{P}}\mathrm{QE}(Q_{\mathrm{SAFE}})$ *is* NP*-complete for OpenPDBs in data complexity. This implies that* $\overline{\mathrm{P}}\mathrm{QE}(\forall\mathrm{FO}_\mathrm{s})$ *is* NP*-complete, and, by duality,* $\underline{\mathrm{P}}\mathrm{QE}(\exists\mathrm{FO}_\mathrm{s})$ *is* coNP*-complete, for OpenPDBs in data complexity.*

The membership results are rather straight-forward. $\overline{\mathrm{P}}\mathrm{QE}(\mathrm{FO}_\mathrm{s})$ can be decided by a nondeterministic Turing machine in polynomial time: given an OpenPDB, and a query $Q \in \mathrm{FO}_\mathrm{s}$, we can guess a completion (which is a PDB), and, based on this completion, we can verify whether the probability of the query exceeds a given value p in polynomial time in data complexity (since the query is assumed to be safe). By similar arguments, we can also conclude that $\underline{\mathrm{P}}\mathrm{QE}(\mathrm{FO}_\mathrm{s})$ is in coNP. These membership results then also apply to ∃FO and ∀FO queries.

Theorem 6.5 additionally shows that there are some queries, which are safe for PDBs, but become hard for OpenPDBs. Briefly, once negation is allowed, it is not always easy to determine the completion upon which the maximal (resp., minimal) probability can be computed. We choose a $\forall\mathrm{FO}_\mathrm{s}$ query, and show that this query is safe in PDBs, but it is NP-hard in OpenPDBs via a reduction from satisfiability of propositional 3CNF formulas. This result is quite intricate, as the class of safe queries enjoy properties which make them easy to compute in PDBs, and, in many cases, the same properties allow us to locally optimize our choices for the bounds in OpenPDBs, i.e., to choose the right completion while avoiding a combinatorial blow-up. Therefore, it is a non-trivial task to identify a query which is safe for PDBs, and, at the same time, hard for OpenPDBs. The full proof involves a series of transformations in order to define such a query. The main computational difference between the two different data models is in the application of the inclusion-exclusion principle: while we obtain cancellations in PDBs, which make the computation of the resulting terms easy, this is not the case for OpenPDBs, mainly due to interacting choices.

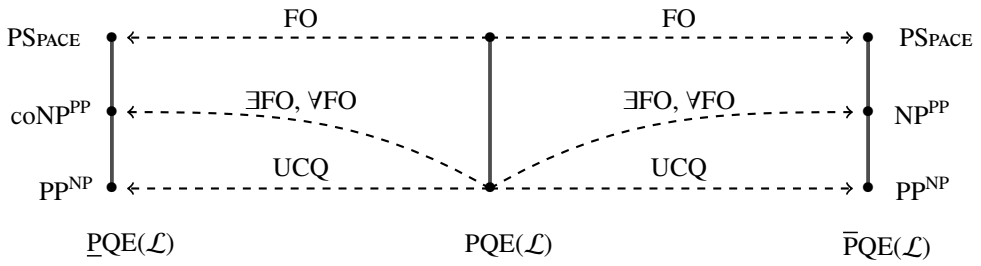


Figure 4: Bounded-arity complexity of $\overline{\text{P}}\text{QE}(\mathcal{L})$ and $\underline{\text{P}}\text{QE}(\mathcal{L})$ for OpenPDBs in comparison to $\text{PQE}(\mathcal{L})$ for PDBs. The given results cover query languages $\mathcal{L} \in \{\text{UCQ}, \exists\text{FO}, \forall\text{FO}, \text{FO}\}$.

We extend our analysis to more general queries, e.g., the class of queries ∃FO, ∀FO, or FO. How can an arbitrary query (i.e., potentially unsafe for PDBs) from these classes be evaluated in OpenPDBs? The following theorem states our results for these query languages.

Theorem 6.6. *Let* $\mathcal{L} \in \{\exists\text{FO}, \forall\text{FO}, \text{FO}\}$. *Then,* $\overline{\text{P}}\text{QE}(\mathcal{L})$ *is* NP^{PP}*-complete, and* $\underline{\text{P}}\text{QE}(\mathcal{L})$ *is* coNP^{PP}*-complete for OpenPDBs in data complexity.*

The membership results follow from similar ideas as before: guess a completion and verify whether the probability of a given query exceeds a threshold relative to this completion. Differently, this verification step now requires a PP oracle, since probabilistic query evaluation over these query classes are PP-complete, as shown in Theorem 6.4. This implies that $\overline{\text{P}}\text{QE}(\text{FO})$ can be decided in NP^{PP}, since we can guess a completion, and based on this completion, we can decide whether the query probability exceeds a threshold, by calling a PP oracle. By analogous arguments, we conclude that $\underline{\text{P}}\text{QE}(\text{FO})$ is in coNP^{PP}. These upper bounds clearly apply to ∃FO and ∀FO queries.

For the hardness results, it is important to note that, unlike in Theorem 6.5, we now have the liberty to choose an unsafe query. That is, even after identifying the right completion, we still need to make a probabilistic computation, which is PP-hard. We make use of this fact to prove the respective hardness results. The first reduction is from an NP^{PP}-complete problem: given a quantified Boolean formula of the form $\Phi = \exists x_1, \ldots, x_\ell\, \mathsf{C}^c\, y_1, \ldots, y_m\, \phi$, where C represents the *counting quantifier* and $\phi = \phi_1 \wedge \cdots \wedge \phi_k$ is a propositional formula in 3CNF, defined over the variables $x_1, \ldots, x_\ell$, $y_1, \ldots, y_m$, decide the validity of Φ. This problem is NP^{PP}-complete [33]. Intuitively, our construction ensures that choosing the maximal completion for an OpenPDB, corresponds to finding a partial assignment to the variables $x_1, \ldots, x_\ell$ in Φ, which can be extended to at least c satisfying assignments. We use a variant of this problem to obtain the respective coNP^{PP}-hardness result.

7. Combined Complexity Results

In the context of databases, the study of combined complexity is less popular, as data complexity often captures the real-world complexity of the relevant problems in a more adequate manner. On the other hand, it is not hard to imagine scenarios where a safe query (in data complexity) could require super-polynomial time in the query. Similar observations motivated some work on this subject; see e.g. [57] where the goal is to isolate cases where probabilistic query evaluation is tractable in combined complexity. We therefore expand our analysis to the combined complexity of probabilistic query evaluation for both PDBs and OpenPDBs. Importantly, we make the *bounded-arity* assumption, i.e., all relations are at most of arity k for some fixed k. Let us first give an overview of the combined complexity results.

7.1. Overview of the Combined Complexity Results

There is yet another subtle reason for (mostly) abandoning combined complexity analysis in PDBs, which is of a technical nature: most of the existing data complexity results (including the data complexity dichotomy) are shown under Turing reductions, which leads to the collapse of many interesting classes, that could make a difference in the case of combined complexity. Our combined complexity analysis, as many other results in this work except from

dichotomy results, are under many-one reductions, which allows us to obtain more fine-grained characterizations. All bounded-arity combined complexity results are summarized in Figure 4.

Arguably, the most interesting result for PDBs is to show that the problem PQE(UCQ) is PP^{NP}-complete in bounded-arity combined complexity, while it is PP-complete in the data complexity. The standard query evaluation problem is already NP-hard in bounded-arity combined complexity for UCQs. Intuitively, this implies that for every world, we need a verification step that is NP-hard. If we restrict our attention to acyclic queries, however, the query evaluation can be done in polynomial time (and even better, see e.g. [58]), and so, probabilistic query evaluation for acyclic conjunctive queries remains in PP. Similarly, query evaluation is PSpace-complete for arbitrary FO queries (where the quantifier nesting is not necessarily bounded). It is easy to see that this class dominates the probabilistic query evaluation problem. All results given for PDBs also hold for combined complexity, i.e., if we remove the bounded-arity assumption.

The results for OpenPDBs can be summarized as follows. First of all, the bounded-arity combined complexity results for OpenPDBs coincide with PDBs when we consider UCQs. This is due to the same reason as in data complexity: for UCQs, we can efficiently reduce upper and lower probabilistic query evaluation in OpenPDBs to probabilistic query evaluation in PDBs. For query languages ∃FO and ∀FO. We have already shown (co)NP^{PP}-hardness results for the data complexity, which clearly also apply to bounded-arity combined complexity. However, these problems do not become harder, despite the fact that query evaluation relative to these queries (which is required for verification) is harder in bounded-arity combined complexity. Intuitively, this holds, since the complexity classes NP^{PP}, and coNP^{PP} are strong enough to do these harder verifications.

7.2. Derivation of the Combined Complexity Results

We now focus on individual combined complexity results, and start our analysis with PDBs. Our first result shows that probabilistic query evaluation for the class of queries UCQ, ∃FO, ∀FO is complete for PP^{NP} for PDBs in combined complexity.

Theorem 7.1. PQE(UCQ), PQE(∃FO), *and* PQE(∀FO) *is* PP^{NP}*-complete for PDBs in bounded-arity combined complexity. These complexity bounds also hold without the bounded-arity assumption.*

The membership results follow from similar ideas to those given for Theorem 6.4, i.e., coding each world induced by a PDB into a nondeterministic Turing machine such that the majority of its runs answer yes if the given instance of probabilistic query evaluation has a positive answer. The main difference is that the complexity of query evaluation on standard databases (i.e., the verification step) is harder in combined complexity: determining whether a database (induced by a given PDB) satisfies the query is NP-complete for ∃FO queries (and UCQs), and coNP-complete for ∀FO queries. Hence, we additionally require calls to an NP oracle to do such verifications.

For hardness, we show that PQE(UCQ) is PP^{NP}-hard, by giving a reduction from the problem of deciding validity of formulas of the form $\Phi = \mathsf{C}^c\, x_1, \ldots, x_m \exists y_1, \ldots, y_n\, \phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_k$, where every ϕ_i is a propositional clause over $x_1, \ldots, x_m, y_1, \ldots, y_n$, and $k, m, n \geq 1$ [33]. Importantly, all the predicates used in the proof are of a bounded arity; that is, the proof applies to the *bounded-arity* combined complexity. The key reason for this hardness is again due to the query evaluation problem for UCQs being NP-complete on standard databases. In fact, if we restrict our attention to e.g. acyclic conjunctive queries, for which query evaluation problem is in polynomial time, probabilistic query evaluation for these queries will remain in PP in combined complexity.

It only remains to determine the complexity of FO queries for PDBs in combined complexity. We obtain the following result for probabilistic query evaluation over FO queries for PDBs.

Theorem 7.2. PQE(FO) *is* PSpace*-complete for PDBs in bounded-arity combined complexity. This result also holds without the bounded-arity assumption.*

Observe that PSpace-hardness of PQE(FO) is immediate since query evaluation is PSpace-complete in standard databases for FO queries (and a database can be viewed as a special PDB). The argument for membership is to walk through all worlds induced by a PDB (each of which is of polynomial size), and sum out their probabilities, if they satisfy the query (which can be determined in polynomial space). This concludes our analysis for PDBs in combined complexity.

We extend our analysis to OpenPDBs in bounded-arity combined complexity. Our first result is for UCQs, which coincides with the bounded-arity combined complexity results given for PDBs.

Theorem 7.3. $\underline{\text{P}}$QE(UCQ) *and* $\overline{\text{P}}$QE(UCQ) *is* PP^{NP}*-complete for OpenPDBs in bounded-arity combined complexity.*

This result follows again by the fact that the satisfaction relation is monotone for UCQs, which allows us to efficiently determine the maximal (resp., minimal) completion for a given OpenPDB, and reduce the problem to probabilistic query evaluation in PDBs. The only subtlety is that each such completion must be of polynomial size, and this is ensured by the fact that the maximal arity of relations is fixed.

Looking into ∃FO and ∀FO queries, we observe a somewhat more interesting phenomenon: the bounded-arity combined complexity of the studied problems coincides with the data complexity of the respective problems.

Theorem 7.4. *Let* $\mathcal{L} \in \{\exists\text{FO}, \forall\text{FO}\}$*. Then,* $\overline{\text{P}}\text{QE}(\mathcal{L})$ *is* NP^{PP}*-complete, and* $\underline{\text{P}}\text{QE}(\mathcal{L})$ *is* coNP^{PP}*-complete for Open-PDBs in bounded-arity combined complexity.*

It may appear somewhat surprising that these results coincide with the data complexity, since the verification step, which requires query evaluation is hard in combined complexity, as discussed earlier. Intuitively, we can decide $\overline{\text{P}}\text{QE}(\exists\text{FO})$ in $\mathfrak{C}^{\text{NP}}$, where $\mathfrak{C} = \text{NP}^{\text{PP}}$, by applying similar ideas to those in Theorem 6.6, and, by additionally calling an NP oracle. It is then sufficient to note that NP^{PP} does not gain additional computational power by calling another NP oracle [31]. Similar arguments then also apply to $\overline{\text{P}}\text{QE}(\exists\text{FO})$, and other cases, yielding the membership results. Hardness results are an immediate consequence of Theorem 6.6, i.e., the respective hardness results given for data complexity.

Finally, as for PDBs, the complexity of classical query evaluation dominates the complexity of probabilistic query evaluation in OpenPDBs.

Theorem 7.5. $\underline{\text{P}}$QE(FO) *and* $\overline{\text{P}}$QE(FO) *are* PSpace*-complete for OpenPDBs in bounded-arity combined complexity.*

The study of combined complexity without the bounded arity assumption is a somewhat intricate notion in Open-PDBs in the following sense: since neither the arity nor the schema is fixed, a completion can grow exponentially, leading to a very high complexity, i.e., we need to perform probabilistic inference over exponentially large completions. These are beyond the focus of our work. We note, however, that a similar observation is also valid for other representations such as Markov Logic Networks, i.e., the size of each world is exponential if the arity of the predicates is not fixed, and with this remark, we conclude our complexity analysis.

8. Related Work

The management of uncertain and probabilistic data is an important problem in many applications of artificial intelligence, e.g., data integration from diverse sources, predictive and stochastic modeling, applications based on (error-prone) sensor readings, and also for automated knowledge base construction [1, 2, 3, 4, 5, 6]. The most basic data model for managing large uncertain data is that of probabilistic databases [14]. Probabilistic database literature is rich, as it is almost as old as traditional database research. We note that the first formulation of possible world semantics in the context of databases is due to Imilieski and Lipski [59], and the work of Fuhr and Röllecke [60] has been very influential in probabilistic database research. For a detailed historical treatment, we refer the reader to standard texts in the literature [14]; here, we mostly focus on recent advancements in probabilistic data and knowledge bases, for which more details can be found in the recent surveys [38, 61].

It is well-known that query evaluation in probabilistic databases is a computationally demanding task, which motivated a line of research aiming at fine-grained classification results. The first thorough study in the database literature appears in the context of reliability analysis for queries, by Grädel, Gurevich and Hirsh [62], where, for instance, the query $\mathsf{C}(x) \wedge \mathsf{R}(x, y) \wedge \mathsf{C}(y)$ is shown to be #P-hard. Dalvi and Suciu obtained the *small dichotomy result* on queries without self-joins [51], and eventually the complete dichotomy on UCQs [50]. There are results which support first-order queries in probabilistic databases [63], but for queries with negation, only partial dichotomy results are known [56]. Other dichotomy results extend the dichotomy for unions of conjunctive queries in various directions; e.g., allowing for disequality ($\neq$) joins in the queries [64], or allowing for inequality ($<$) joins in the queries [65]. A trichotomy result is given for queries with aggregation [66]. Amarilli and Ceylan [67] recently extended the dichotomy for UCQs to *infinite* unions of conjunctive queries over binary signatures. This implies a dichotomy for a large class of query languages beyond UCQs, including negation-free (disjunctive) Datalog, regular path queries, and a large class of ontology-mediated queries on binary signatures.

Our work is motivated by open-world reasoning. The open-world assumption is common in deterministic knowledge bases, which are widely studied in the context of ontology languages, mostly based on description logics [68], or Datalog$^{\pm}$ [69]. Instead of posing the queries directly to the database, the idea is to use a logical theory (or ontology) as a query interface, in order to obtain a more complete set of answers from an incomplete knowledge base via the open-world assumption. This paradigm is known as ontology-mediated query answering [70], and the open-world assumption is a driving force for ontology-based technologies. The literature on probabilistic extensions of ontology languages is rich, and ontology-mediated queries for probabilistic databases have been investigated in the context of both description logics [71, 72] and Datalog$^{\pm}$ [73, 74, 75, 40]. Importantly, these models are typically open-domain, i.e., they allow reasoning over infinitely many objects in the domain (unlike our finite-domain assumption). There is a key restriction in such models that allows to preserve some nice computational results: although reasoning is over infinitely many objects, the query semantics is defined relative to probability distributions over known atoms in the database to ensure that the probability space remains finite. One subtle aspect is that these models employ certain answer semantics, and so if a query is not entailed, its probability is set to zero [71, 72, 74]. That is, their semantics partially import closed-world PDB semantics for non-entailments. Other models[73, 40] employ a semantics closely related to Markov logic networks, while keeping the open-domain reasoning. We note that OpenPDBs are extended with ontological knowledge [75] to allow for probabilities from a default interval for non-entailments, while the ontology allows for open-domain reasoning. For further details on the semantic differences, such as open-domain vs closed-domain, open-world vs closed-world models; we refer the reader to [61], where a detailed classification, including OpenPDBs, is given.

Our work draws inspirations from *lifted inference* [76] in avoiding explicit reasoning over all atoms, or constants: our lifted inference algorithm operates on first-order structures to exploit symmetries thereby avoiding a complete grounding. In this respect, our work brings together the high-level reasoning of lifted inference and the data-centric reasoning of probabilistic databases. OpenPDBs are also closely related to credal networks [48]. The major difference is that one source of hardness for probabilistic inference in credal networks is due to the conditional dependencies encoded in the network structure, which is very different from OpenPDBs, as such dependencies stem from the query in OpenPDBs. Work in probabilistic logic programming has studied their complexity for different semantics, including credal semantics [77]. OpenPDBs also motivated further research to extend the open-world probabilistic database model to have schema-level constraints on completion probabilities [78]. OpenPDBs are defined over a finite domain, and the work of Grohe and Lindner [79] extends the open-world probabilistic database model to infinite universes.

Weighted model counting (WMC) has emerged as a unifying approach for probabilistic inference in various data models [80, 81]. Query answering in probabilistic databases reduces to WMC over DNF structures, as every conjunctive query is equivalent to a DNF via its lineage representation (which is of polynomial size in data complexity). WMC over DNFs is clearly #P-hard [27], which motivated two paradigms for solving this problem. One prominent approach for WMC is based on knowledge compilation [82, 83], which compiles the problem in a target language, upon which the respective task is tractable. That is, the computational overhead is pushed to an offline phase, amortized by a large number of online queries. Knowledge compilation has been studied also in the context of PDBs [84]. Another prominent approach for WMC is approximate solving, which provides approximations of the model count as opposed to an exact solution. There are numerous approximation algorithms for weighted model counting: a classical result from Karp, Luby and Madras [85] asserts that weighted model counting over DNF structures admits a *fully polynomial randomized approximation scheme* (FPRAS). Hashing-based approximation techniques [86] can solve the unweighted model counting problem on DNF structures with probabilistic accuracy guarantees. These algorithms are not very scalable in practice; recently, a neural model counting approach has been proposed [87] for fast weighted model counting, which does not provide accuracy guarantees, but experiments suggest a reliable prediction accuracy.

Many probabilistic relational database management systems, dedicated for large-scale probabilistic data processing, have been developed, such as MystiQ [88] and SPROUT [89]. Importantly, SPROUT [89] also supports a type of open-world inference in the sense that it enables querying "Google Squared" tables which are extracted from open-world text without a fixed vocabulary. Other approaches include Slimshot [39] which can encode complex relations over a closed-domain, and Tuffy [90] which uses Markov chain Monte Carlo for probabilistic inference.

Summary and Outlook

We proposed a probabilistic data model, called OpenPDBs, that acknowledges the incomplete nature of the knowledge bases, as part of its semantics. In OpenPDBs, atoms that are not in the database still remain possible with some default probability. This is in contrast with PDBs, where atoms that do not appear in the database are assigned a probability zero by the CWA. Our work builds on the foundations of tuple-independent PDBs [14]. In particular, we extend the dichotomy result of Dalvi and Suciu [50], given for UCQs to OpenPDBs. As a side contribution, we observe that the original dichotomy for PDBs is stronger: under reasonable assumptions, all safe queries can be computed in linear time. We also show that nonmonotone queries are typically harder in OpenPDBs than those in PDBs. Our analysis includes both data and combined complexity and provides a complete picture for the complexity landscape of OpenPDBs in comparison with PDBs.

OpenPDBs already motivated several lines of work. One of the key challenges in OpenPDBs is to restrict the open world to provide tighter probability bounds, as the default probability interval may not always be very informative. One way of excluding spurious possible worlds, and limiting the probability mass of open atoms is by using an additional knowledge representation layer towards more informative probability bounds [75]. An alternative way is to define schema-level constraints on the probability space, ensuring more informative bounds [78]. Our study focuses on finite domains, which may not be satisfactory in every application domain, which motivated an extension to infinite universes [79].

The focus of our work is merely on exact inference (of the associated decision problems). It is well-known that probabilistic query evaluation admits an FPRAS for UCQs, as it can be reduced to weighted model counting over DNF structures. This results immediately translates to OpenPDBs, by the reductions presented in this paper. We think that a dedicated study for approximate inference in OpenPDBs can be an interesting direction for future work.

Acknowledgments

This work is partially supported by the UK EPSRC grant EP/R013667/1, NSF grants #IIS-1943641, #IIS-1633857, #CCF-1837129, DARPA XAI grant #N66001-17-2-4032, a UCLA Samueli Fellowship, and gifts from Intel and Facebook Research.

Bibliography

[1] T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, J. Welling, Never-Ending Learning, in: Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI-15), AAAI Press, 2015, pp. 2302–2310.

[2] J. Shin, S. Wu, F. Wang, C. De Sa, C. Zhang, C. Ré, Incremental knowledge base construction using deepdive, Proceedings of VLDB Endowment 8 (11) (2015) 1310–1321. doi:10.14778/2809974.2809991.

[3] A. Fader, S. Soderland, O. Etzioni, Identifying relations for open information extraction, in: Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP-11), Association for Computational Linguistics, 2011, p. 15351545.

[4] J. Hoffart, F. M. Suchanek, K. Berberich, G. Weikum, Yago2: A spatially and temporally enhanced knowledge base from wikipedia, Artificial Intelligence 194 (2013) 28–61. doi:10.1016/j.artint.2012.06.001.

[5] W. Wu, H. Li, H. Wang, K. Q. Zhu, Probase: A probabilistic taxonomy for text understanding, in: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, Association for Computing Machinery, 2012, pp. 481–492. doi:10.1145/2213836.2213891.

[6] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, W. Zhang, 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, Association for Computing Machinery, 2014, pp. 601–610. doi:10.1145/2623330.2623623.

[7] J. P. Ku, J. L. Hicks, T. Hastie, J. Leskovec, C. Ré, S. L. Delp, 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) (2015) 1120–1125. doi:10.1093/jamia/ocv071.

[8] S. E. Peters, C. Zhang, M. Livny, C. Ré, A Machine Reading System for Assembling Synthetic Paleontological Databases., PLoS ONE 9 (12). doi:10.1371/journal.pone.0113523.

[9] M. Mintz, S. Bills, R. Snow, D. Jurafsky, Distant supervision for relation extraction without labeled data, in: Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP, Association for Computational Linguistics, 2009, pp. 1003–1011.

[10] A. Bordes, J. Weston, R. Collobert, Y. Bengio, Learning structured embeddings of knowledge bases, in: W. Burgard, D. Roth (Eds.), Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI-11), AAAI Press, 2011, p. 301306.

[11] R. Socher, D. Chen, C. D. Manning, A. Ng, Reasoning with neural tensor networks for knowledge base completion, in: Proceedings of the 26th International Conference on Neural Information Processing Systems, Curran Associates, Inc., 2013, pp. 926–934.
[12] W. Y. Wang, K. Mazaitis, W. W. Cohen, Programming with personalized pagerank: A locally groundable first-order probabilistic logic, in: Proceedings of the 22nd ACM International Conference on Information & Knowledge Management (CIKM-13), Association for Computing Machinery, 2013, pp. 2129–2138. doi:10.1145/2505515.2505573.
[13] L. De Raedt, A. Dries, I. Thon, G. Van den Broeck, M. Verbeke, Inducing probabilistic relational rules from probabilistic examples, in: Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI-15), AAAI Press, 2015, p. 18351843.
[14] D. Suciu, D. Olteanu, C. Ré, C. Koch, Probabilistic Databases, Vol. 3, Morgan & Claypool Publishers, 2011. doi:https://doi.org/10.2200/S00362ED1V01Y201105DTM016.
[15] L. Galárraga, S. Razniewski, A. Amarilli, F. M. Suchanek, Predicting completeness in knowledge bases, in: Proceedings of the 10th ACM International Conference on Web Search and Data Mining (WSDM-17), Association for Computing Machinery, 2017, pp. 375–383. doi:10.1145/3018661.3018739.
[16] R. Reiter, On closed world data bases, Springer US, 1978, pp. 55–76. doi:10.1007/978-1-4684-3384-5_3.
[17] C. M. Bishop, Pattern recognition and machine learning, Springer-Verlag, 2006.
[18] I. Levi, The Enterprise of Knowledge: : An Essay on Knowledge, Credal Probability, and Chance, MIT Press, 1980.
[19] İ. İ. Ceylan, A. Darwiche, G. Van den Broeck, Open-world probabilistic databases, in: Proceedings of the 15th International Conference on Principles of Knowledge Representation and Reasoning (KR-16), AAAI Press, 2016, pp. 339–348.
[20] İ. İ. Ceylan, A. Darwiche, G. Van den Broeck, Open-world probabilistic databases: An abridged report, in: Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI-17), 2017, pp. 4796–4800. doi:10.24963/ijcai.2017/669.
[21] İ. İ. Ceylan, Query answering in probabilistic data and knowledge bases, Ph.D. thesis, TU Dresden (2017).
[22] T. Hinrichs, M. Genesereth, Herbrand logic, Tech. Rep. LG-2006-02, Stanford University (2006).
[23] E. F. Codd, Relational completeness of data base sublanguages, IBM Corporation, 1972.
[24] S. Abiteboul, R. Hull, V. Vianu (Eds.), Foundations of Databases: The Logical Level, 1st Edition, Addison-Wesley Longman Publishing Co., Inc., 1995.
[25] L. Libkin, Elements of Finite Model Theory, Springer-Verlag, 2004.
[26] M. Sipser, Introduction to the Theory of Computation, 1st Edition, International Thomson Publishing, 1996.
[27] L. G. Valiant, The complexity of computing the permanent, Theoretical Computer Science 8 (2) (1979) 189–201. doi:https://doi.org/10.1016/0304-3975(79)90044-6.
[28] J. T. Gill, Computatonal complexity of probabilistic Turing machines, SIAM Journal on Computing 6 (4) (1977) 675–695. doi:10.1137/0206049.
[29] M. L. Littman, S. M. Majercik, T. Pitassi, Stochastic boolean satisfiability, Journal of Automated Reasoning 27 (3) (2001) 251–296. doi:10.1023/A:1017584715408.
[30] R. Beigel, N. Reingold, D. Spielman, PP is closed under intersection, Journal of Computer and System Sciences 50 (2) (1995) 191–202. doi:https://doi.org/10.1006/jcss.1995.1017.
[31] S. Toda, O. Watanabe, Polynomial-time 1-Turing reductions from #PH to #P, Theoretical Computer Science 100 (1) (1992) 205–221. doi:https://doi.org/10.1016/0304-3975(92)90369-Q.
[32] S. Toda, On the computational power of pp and +p, in: Proceedings of the 30th Annual Symposium on Foundations of Computer Science, 1989, pp. 514–519. doi:10.1109/SFCS.1989.63527.
[33] K. W. Wagner, The complexity of combinatorial problems with succinct input representation, Acta Informatica 23 (3) (1986) 325–356. doi:10.1007/BF00289117.
[34] M. L. Littman, J. Goldsmith, M. Mundhenk, The computational complexity of probabilistic planning, Journal of Artificial Intelligence Research 9 (1998) 1–36. doi:10.1613/jair.505.
[35] J. D. Park, A. Darwiche, Complexity results and approximation strategies for MAP explanations, Journal of Artificial Intelligence Research 21 (1) (2004) 101–133. doi:10.1613/jair.1236.
[36] S. A. Cook, The Complexity of Theorem-proving Procedures, in: Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC-71), Association for Computing Machinery, 1971, pp. 151–158. doi:10.1145/800157.805047.
[37] M. Y. Vardi, The complexity of relational query languages, in: H. R. Lewis, B. B. Simons, W. A. Burkhard, L. H. Landweber (Eds.), Proceedings of the 14th Annual ACM Symposium on Theory of Computing (STOC-82), Association for Computing Machinery, 1982, pp. 137–146. doi:10.1145/800070.802186.
[38] G. Van den Broeck, D. Suciu, Query processing on probabilistic data: A survey, Foundations and Trends in Databases 7 (3/4) (2017) 197–341. doi:10.1561/1900000052.
[39] E. Gribkoff, D. Suciu, SlimShot: In-Database Probabilistic Inference for Knowledge Bases, Proceedings of VLDB Endowment 9 (7) (2016) 552563. doi:10.14778/2904483.2904487.
[40] S. Borgwardt, İ. İ. Ceylan, T. Lukasiewicz, Ontology-mediated query answering over log-linear probabilistic data, in: Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI-19), 2019, pp. 2711–2718. doi:https://doi.org/10.1609/aaai.v33i01.33012711.
[41] T. Sato, A statistical learning method for logic programs with distribution semantics, in: Proceedings of the 12th International Conference on Logic Programming (ICLP-95), MIT Press, 1995, pp. 715–729.
[42] D. Poole, The independent choice logic for modelling multiple agents under uncertainty, Artificial Intelligence 94 (1-2) (1997) 7–56. doi:10.1016/S0004-3702(97)00027-1.
[43] L. De Raedt, A. Kimmig, H. Toivonen, ProbLog: A probabilistic prolog and its application in link discovery, in: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07), Morgan Kaufmann, 2007, pp. 2468–2473.
[44] C. Sutton, A. McCallum, An introduction to conditional random fields, Machine Learning 4 (4) (2011) 267–373. doi:10.1561/2200000013.
[45] L. A. Galárraga, C. Teflioudi, K. Hose, F. Suchanek, Amie: Association rule mining under incomplete evidence in ontological knowledge

bases, in: Proceedings of the 22nd International Conference on World Wide Web (WWW-13), Association for Computing Machinery, 2013, p. 413422. doi:10.1145/2488388.2488425.
[46] R. Munroe, Google's datacenters on punch cards (2015).
[47] J. Y. Halpern, Reasoning about uncertainty, MIT Press, 2003.
[48] F. G. Cozman, Credal networks, Artificial Intelligence 120 (2) (2000) 199–233. doi:https://doi.org/10.1016/S0004-3702(00)00029-1.
[49] R. Reiter, A logic for default reasoning, Artificial Intelligence 13 (1) (1980) 81–132. doi:https://doi.org/10.1016/0004-3702(80)90014-4.
[50] N. Dalvi, D. Suciu, The dichotomy of probabilistic inference for unions of conjunctive queries, Journal of ACM 59 (6) (2012) 1–87. doi:10.1145/2395116.2395119.
[51] N. Dalvi, D. Suciu, Efficient query evaluation on probabilistic databases, The VLDB Journal 16 (4) (2007) 523–544. doi:10.1007/s00778-006-0004-3.
[52] E. Gribkoff, G. Van den Broeck, D. Suciu, 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), AUAI Press, 2014, pp. 280–289.
[53] Y. Sagiv, M. Yannakakis, Equivalences among relational expressions with the union and difference operators, Journal of ACM 27 (4) (1980) 633–655. doi:10.1145/322217.322221.
[54] C. P. De Campos, F. G. Cozman, The inferential complexity of bayesian and credal networks, in: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05), 2005, pp. 1313–1318.
[55] A. Darwiche, Modeling and Reasoning with Bayesian Networks, Cambridge University Press, 2009. doi:10.1111/j.1751-5823.2012.00179_15.x.
[56] R. Fink, D. Olteanu, Dichotomies for queries with negation in probabilistic databases, ACM Transactions on Database Systems (TODS) 41 (1) (2016) 4:1–4:47. doi:10.1145/2877203.
[57] A. Amarilli, M. Monet, P. Senellart, Conjunctive queries on probabilistic graphs: Combined complexity, in: Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS-17), Association for Computing Machinery, 2017, pp. 217–232. doi:10.1145/3034786.3056121.
[58] G. Gottlob, N. Leone, F. Scarcello, The complexity of acyclic conjunctive queries, Journal of ACM 48 (3) (2001) 431–498. doi:10.1145/382780.382783.
[59] T. ImieliÅski, W. Lipski, Incomplete information in relational databases, in: J. Mylopolous, M. Brodie (Eds.), Readings in Artificial Intelligence and Databases, Morgan Kaufmann, 1989, pp. 342 – 360. doi:https://doi.org/10.1016/B978-0-934613-53-8.50027-3.
[60] N. Fuhr, T. Rölleke, A probabilistic relational algebra for the integration of information retrieval and database systems, ACM Transactions on Information Systems (TOIS) 15 (1) (1997) 32–66. doi:10.1145/239041.239045.
[61] S. Borgwardt, İ. İ. Ceylan, T. Lukasiewicz, Recent advances in querying probabilistic knowledge bases, in: Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI-18), 2018, pp. 5420–5426. doi:10.24963/ijcai.2018/765.
[62] E. Grädel, Y. Gurevich, C. Hirsch, The complexity of query reliability, in: Proceedings of the 17th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS-98), Association for Computing Machinery, 1998, pp. 227–234. doi:10.1145/275487.295124.
[63] R. Fink, D. Olteanu, S. Rath, Providing support for full relational algebra in probabilistic databases, in: Proceedings of the 27th International Conference on Data Engineering (ICDE-11), 2011, pp. 315–326. doi:10.1109/ICDE.2011.5767912.
[64] D. Olteanu, J. Huang, Using obdds for efficient query evaluation on probabilistic databases, in: Proceedings of the 2nd International Conference on Scalable Uncertainty Management (SUM-08), Vol. 5291 of Lecture Notes in Computer Science, Springer-Verlag, 2008, pp. 326–340.
[65] D. Olteanu, J. Huang, Secondary-storage confidence computation for conjunctive queries with inequalities, in: Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data, Association for Computing Machinery, 2009, pp. 389–402. doi:10.1145/1559845.1559887.
[66] C. Ré, D. Suciu, The trichotomy of having queries on a probabilistic database, The VLDB Journal 18 (5) (2009) 1091–1116.
[67] A. Amarilli, İ. İ. Ceylan, A dichotomy for homomorphism-closed queries on probabilistic graphs, in: C. Lutz (Ed.), Proceedings of the 23rd International Conference on Database Theory (ICDT-20), Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2020.
[68] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider (Eds.), The Description Logic Handbook: Theory, Implementation, and Applications, 2nd Edition, Cambridge University Press, 2007.
[69] A. Calì, G. Gottlob, T. Lukasiewicz, A general Datalog-based framework for tractable query answering over ontologies, Journal of Web Semantics 14 (2012) 57–83. doi:https://doi.org/10.1016/j.websem.2012.03.001.
[70] M. Bienvenu, B. T. Cate, C. Lutz, F. Wolter, Ontology-based data access: A study through disjunctive datalog, csp, and mmsnp, ACM Transactions on Database Systems (TODS) 39 (4) (2014) 1–44.
[71] J. C. Jung, C. Lutz, Ontology-based access to probabilistic data with owl ql, in: Proceedings of the 11th International Conference on The Semantic Web - Volume Part I, Springer-Verlag, 2012, pp. 182–197.
[72] İ. İ. Ceylan, R. Peñaloza, Probabilistic query answering in the bayesian description logic BEL, in: Proceedings of the 9th International Conference on Scalable Uncertainty Management (SUM-15), Vol. 9310, Springer-Verlag, 2015, pp. 21–35.
[73] G. Gottlob, T. Lukasiewicz, M. V. Martinez, G. I. Simari, Query answering under probabilistic uncertainty in datalog+/- ontologies, Annals of Mathematics and Artificial Intelligence 69 (1) (2013) 37–72. doi:10.1007/s10472-013-9342-1.
[74] İ. İ. Ceylan, T. Lukasiewicz, R. Peñaloza, Complexity results for probabilistic datalog±, in: Proceedings of the 28th European Conference on Artificial Intelligence (ECAI-16), Vol. 285, IOS Press, 2016, pp. 1414–1422. doi:10.3233/978-1-61499-672-9-1414.
[75] S. Borgwardt, İ. İ. Ceylan, T. Lukasiewicz, Ontology-mediated queries for probabilistic databases, in: Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI-17), AAAI Press, 2017, pp. 1063–1069.
[76] D. Poole, First-order probabilistic inference, in: Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03), Vol. 3, 2003, pp. 985–991.

[77] F. G. Cozman, D. D. Mauá, On the semantics and complexity of probabilistic logic programs, Journal of Artificial Intelligence Research 60 (1) (2017) 221262.
[78] T. Friedman, G. Van den Broeck, On constrained open-world probabilistic databases, in: Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI-19), 2019, pp. 5722–5729. doi:10.24963/ijcai.2019/793.
[79] M. Grohe, P. Lindner, Probabilistic databases with an infinite open-world assumption, in: Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS-19), 2019, pp. 17–31. doi:10.1145/3294052.3319681.
[80] C. P. Gomes, A. Sabharwal, B. Selman, Model counting, in: Handbook of Satisfiability, IOS Press, 2009.
[81] M. Chavira, A. Darwiche, On probabilistic inference by weighted model counting, Artificial Intelligence 172 (6-7) (2008) 772–799. doi:10.1016/j.artint.2007.11.002.
[82] M. Cadoli, F. Donini, A survey on knowledge compilation, AI Communications 10 (3-4) (1997) 137–150.
[83] A. Darwiche, P. Marquis, A Knowledge Compilation Map, Journal of Artificial Intelligence Research 17 (1) (2002) 229–264.
[84] A. Jha, D. Suciu, Knowledge Compilation Meets Database Theory: Compiling Queries to Decision Diagrams, Theory of Computing Systems 52 (3) (2013) 403440. doi:10.1007/s00224-012-9392-5.
[85] R. M. Karp, M. Luby, N. Madras, Monte-Carlo approximation algorithms for enumeration problems, J. Algorithms 10 (3) (1989) 429448. doi:10.1016/0196-6774(89)90038-2.
[86] K. S. Meel, A. A. Shrotri, M. Y. Vardi, On hashing-based approaches to approximate DNF-counting, in: Procceedings of 37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, (FSTTCS-17), 2017, pp. 1–14. doi:10.4230/LIPIcs.FSTTCS.2017.41.
[87] R. Abboud, İ. İ. Ceylan, T. Lukasiewicz, Learning to reason: Leveraging neural networks for approximate DNF counting, in: Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI-20), AAAI Press, 2020.
[88] J. Boulos, N. Dalvi, B. Mandhani, S. Mathur, C. Ré, D. Suciu, Mystiq: A system for finding more answers by using probabilities, in: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, Association for Computing Machinery, 2005, pp. 891–893. doi:10.1145/1066157.1066277.
[89] R. Fink, A. Hogue, D. Olteanu, S. Rath, Sprout^2: A squared query engine for uncertain web data, Association for Computing Machinery, 2011, pp. 1299–1302. doi:10.1145/1989323.1989481.
[90] F. Niu, C. Ré, A. Doan, J. W. Shavlik, Tuffy: Scaling up statistical inference in markov logic networks using an RDBMS, Proceedings of VLDB Endowment 4 (6) (2011) 373–384. doi:10.14778/1978665.1978669.
[91] G. S. Tseitin, On the Complexity of Derivation in Propositional Calculus, Springer-Verlag, pp. 466–483. doi:10.1007/978-3-642-81955-1_28.

Appendix A. Proofs of Semantic Results

This part contains the initial results stated in Section 5, most of which are semantic results.

Proof of Lemma 5.2

It is sufficient to choose an ϵ value, which has strictly lower probability than any of the worlds induced by the given PDB $\mathcal{P}$. Let us denote by n be the number of atoms in the PDB $\mathcal{P}$, and by m the maximal precision among the fact probabilities in $\mathcal{P}$. We set $\epsilon = 0.0\ldots01$, which has exactly $n \cdot m$ zero's, and ensures that (i) ϵ has a lower probability than any of the worlds induced by $\mathcal{P}$, and (ii) the size of ϵ is polynomial in $\mathcal{P}$. It is then easy to verify that $\mathrm{P}_{\mathcal{P}}(Q) > p$ if and only if $\mathrm{P}_{\mathcal{P}}(Q) \geq p + \epsilon$. Conversely, it is easy to verify that $\mathrm{P}_{\mathcal{P}}(Q) \geq p$ if and only if $\mathrm{P}_{\mathcal{P}}(Q) > p - \epsilon$. Analogous arguments hold also for $<$ and $\leq$.

Proof of Corollary 5.3

Let Q be a *safe* UCQ for PDBs. Then, for any PDB $\mathcal{P}$, the exact probability $\mathrm{P}_{\mathcal{P}}(Q)$ can be computed in polynomial time. Hence, it is possible to decide whether $\mathrm{P}_{\mathcal{P}}(Q) > p$ for a given threshold p, in polynomial time. Thus, $\mathrm{PQE}(Q)$ is in P for any *safe* UCQ Q.

Conversely, let Q be an *unsafe* UCQ for PDBs. Then, there exists a PDB $\mathcal{P}$ such that computing $\mathrm{P}(Q)$ is #P-hard *under polynomial-time Turing reductions*. Let us loosely denote by $\mathrm{P}(Q)$ the problem of computing $\mathrm{P}(Q)$. We need to show that PP is contained in $\mathrm{P}^{\mathrm{PQE}(Q)}$, i.e., any problem in PP can be reduced to $\mathrm{PQE}(Q)$ under polynomial time Turing reductions.

To show this, let A be a problem in PP. Let us denote by $\#\mathsf{A}$, its computation problem. By assumption, $\#\mathsf{A}$, 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 $\mathrm{FP}^{\mathrm{PQE}(Q)}$ 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 $\mathfrak{C} = \mathrm{FP}^{\mathrm{PQE}(Q)}$. 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 $\mathrm{PQE}(Q)$ can be queried directly). This shows that A is in $\mathrm{P}^{\mathrm{PQE}(Q)}$, which proves the result.

Proof of Theorem 5.9

First, note that the functions $\overline{\mathrm{P}}_{\mathcal{G}} : \mathrm{FO} \mapsto [0, \lambda]$ and $\underline{\mathrm{P}}_{\mathcal{G}} : \mathrm{FO} \mapsto [0, \lambda]$ are *well-defined* w.r.t. the set $\mathrm{K}_{\mathcal{G}}$, i.e., the existence of a *maximum* (resp., *minimum*) is ensured by the properties of credal sets. We need to show that the maximal (resp., minimal) probabilities of queries can always be obtained from the extreme probability distributions. We prove the claim only for $\overline{\mathrm{P}}$ as $\underline{\mathrm{P}}$ can be treated analogously.

To simplify the proof, we use the lineage representation of the database atoms, which can be realized simply by introducing a propositional literal p_t for every atom t. Similarly, we focus on the lineage of the query, which can be obtained by first grounding the query and then converting it into a propositional formula by replacing every tuple in the ground query with its lineage. It is well-known that every first-order query relative to a finite structure has a corresponding propositional lineage representation and thus our assumption is without loss of generality. Since any propositional formula is equivalent to a formula in 3CNF, we can further assume that the lineage is in 3CNF. Moreover, for simplicity we assume that the CNF contains *exactly* three literals. Thus, it suffices to prove the claim for 3CNF formulas $\phi = c_1 \wedge \ldots \wedge c_n$ where $c_i = (\neg) l_{i_1} \vee (\neg) l_{i_2} \vee (\neg) l_{i_3}$.

Suppose that there is a probability distribution P, where the probabilities of k (positive) literals in ϕ are set to intermediate probability values from the interval $(0, \lambda)$. We prove that, for each such literal l, there is (at least) one extreme assignment to l that does not decrease the probability of ϕ. Formally, given P, we define two new probability distributions $\mathrm{P}^{l=\lambda}$ and $\mathrm{P}^{l=0}$ such that $\mathrm{P}^{l=\lambda}(l_{ij}) = \mathrm{P}(l_{ij})$ and $\mathrm{P}^{l=0}(l_{ij}) = \mathrm{P}(l_{ij})$ for all l_{ij} different from l and $\mathrm{P}^{l=\lambda}(l) = \lambda$, and $\mathrm{P}^{l=0}(l) = 0$.

Claim. Either $\mathrm{P}^{l=\lambda}(\phi) \geq \mathrm{P}(\phi)$, or $\mathrm{P}^{l=0}(\phi) \geq \mathrm{P}(\phi)$ holds.

To prove the claim, suppose that $\mathrm{P}^{l=\lambda}(\phi) < \mathrm{P}(\phi)$, i.e., the probability of $\phi = c_1 \wedge \ldots \wedge c_n$ decreases if we increase the probability of l to λ. We make a case analysis.

Case 1. Assume that the literal l appears only positively in ϕ. This immediately leads to a contradiction since, if for every clause c_i, l appears positively, then the probability of ϕ is clearly *monotone* in l. Thus, $\mathrm{P}^{l=\lambda}(\phi) \geq \mathrm{P}(\phi)$.

Case 2. Assume that the literal l appears only negatively in ϕ. If for every clause c_i, l appears negatively, then the probability of ϕ is *antitone* in l. This immediately implies that $\mathrm{P}^{l=0}(\phi) \geq \mathrm{P}(\phi)$ since further decreasing the probability of l increases the probability of ϕ.

Case 3. Assume that the literal l appears both positively and negatively in ϕ. We can summarize all clauses where l appears positively with the formula

$$l \vee \Big(\underbrace{\big((\neg)u_1 \vee (\neg)u_2\big) \wedge \ldots \wedge \big((\neg)u_r \vee (\neg)u_{r+1}\big)}_{\Delta_1} \Big),$$

and similarly the clauses where l appears negatively with the formula

$$\neg l \vee \Big(\underbrace{\big((\neg)v_1 \vee (\neg)v_2\big) \wedge \ldots \wedge \big((\neg)v_s \vee (\neg)v_{s+1}\big)}_{\Delta_2} \Big).$$

Moreover, let Δ_3 be the conjunction of all clauses where the literal l does not appear. Thus, we obtain $(l \vee \Delta_1) \wedge (\neg l \vee \Delta_2) \wedge \Delta_3$ as a rewriting of ϕ.

We can further simplify ϕ and deduce:

$$\mathrm{P}(\phi) = \mathrm{P}(l)\,\mathrm{P}(\Delta_2 \wedge \Delta_3) + \mathrm{P}(\neg l)\,\mathrm{P}(\Delta_1 \wedge \Delta_3),$$

which is maximized by setting $\mathrm{P}(l) = \lambda$, if $\mathrm{P}(\Delta_2 \wedge \Delta_3) > \mathrm{P}(\Delta_1 \wedge \Delta_3)$, and by setting $\mathrm{P}(l) = 0$, otherwise. This shows that if $\mathrm{P}^{l=\lambda}(\phi) < \mathrm{P}(\phi)$ then $\mathrm{P}^{l=0}(\phi) \geq \mathrm{P}(\phi)$. Therefore, this concludes our case analysis and proves the claim.

Observe that this argument can be applied repeatedly until there is no such literal left, i.e., all literals are assigned a probability value that is extreme. Clearly, this procedure terminates, and implies that, for any probability distribution that is maximal, but not extreme, we can find a corresponding extreme distribution that is also maximal.

Proof of Theorem 5.10

We prove the result for the upper bound: $\overline{\mathrm{P}}_{\mathcal{G}}(Q) = \mathrm{P}_{\mathcal{P}_\lambda}(Q)$. The proof for the lower bound $\underline{\mathrm{P}}_{\mathcal{G}}(Q)$, can be obtained analogously. It is easy to see that the function $\overline{\mathrm{P}}_{\mathcal{G}} : \mathrm{UCQ} \mapsto [0, \lambda]$ is monotone for any choice of λ. By Definition 4.3, we know that $\mathrm{P}_{\mathcal{P}_\lambda} \in \mathrm{K}_{\mathcal{G}}$. Thus, we obtain $\overline{\mathrm{P}}_{\mathcal{G}}(Q) \geq \mathrm{P}_{\mathcal{P}_\lambda}(Q)$. To show the other direction, i.e., $\overline{\mathrm{P}}_{\mathcal{G}}(Q) \leq \mathrm{P}_{\mathcal{P}_\lambda}(Q)$, assume by contradiction that $\overline{\mathrm{P}}_{\mathcal{G}}(Q) > \mathrm{P}_{\mathcal{P}_\lambda}(Q)$. Then, by Theorem 5.9, there exists a PDB $\hat{\mathcal{P}}$ that uses only the extreme points for the open tuples (i.e., induces an extreme distribution) and that satisfies $\mathrm{P}_{\hat{\mathcal{P}}}(Q) = \overline{\mathrm{P}}_{\mathcal{G}}(Q) > \mathrm{P}_{\mathcal{P}_\lambda}(Q)$. Since $\mathcal{P}_\lambda$ and $\hat{\mathcal{P}}$ induce different distributions, there must exist at least one atom t for which $\mathrm{P}_{\hat{\mathcal{P}}}(t) = 0$ and $\mathrm{P}_{\mathcal{P}_\lambda}(t) = \lambda$. Then, by the monotonicity of $\overline{\mathrm{P}}$ on UCQs, it follows that $\mathrm{P}_{\hat{\mathcal{P}}}(Q) \leq \mathrm{P}_{\mathcal{P}_\lambda}(Q)$, which leads to a contradiction.

Proof of Lemma 5.11

This is a simple consequence of the query semantics: either $\mathcal{D} \models Q$ or $\mathcal{D} \models \neg Q$ holds for any database $\mathcal{D}$ and query Q. On this level, the semantics forces completeness, and therefore, it is never the case that neither $\mathcal{D} \models Q$ nor $\mathcal{D} \models \neg Q$ holds. By this argument, for any probability distribution P, it holds that $\mathrm{P}(Q) = 1 - \mathrm{P}(\neg Q)$. Using this and the existence of maximal and minimal distributions in OpenPDBs, it is easy to deduce:

$$1 - \overline{\mathrm{P}}_{\mathcal{G}}(Q) = 1 - \max\{\mathrm{P}(Q) \mid \mathrm{P} \in \mathrm{K}_{\mathcal{G}}\} = \min\{1 - \mathrm{P}(Q) \mid \mathrm{P} \in \mathrm{K}_{\mathcal{G}}\} = \min\{\mathrm{P}(\neg Q) \mid \mathrm{P} \in \mathrm{K}_{\mathcal{G}}\} = \underline{\mathrm{P}}_{\mathcal{G}}(\neg Q),$$

and the other case can be shown analogously.

Appendix B. Proofs of Data Complexity Results

This part contains all the proofs of the data complexity results stated in Section 6.

Proof of Theorem 6.2

Consider an OpenPDB $\mathcal{G} = (\mathcal{P}, \lambda)$ over a sorted domain, and a UCQ. Let us denoted by $\mathcal{G}^r = (\mathcal{P}^r, \lambda)$ the preprocessed OpenPDB, which is ranked. Note that in general $\mathcal{P}^r$ can be polynomially larger than $\mathcal{P}$.

First, we show that the number of calls in the recursion tree of Algorithm 1 is linear in the size of $\mathcal{P}^r$ (i.e., the number of atoms in the preprocessed PDB). We can ignore calls below each invocation of Line 23, as these calls no longer depend on $\mathcal{P}^r$. For the remaining calls to $\textsc{Lift}^{\mathrm{R}}_{\mathrm{O}}$, we show a constant upper bound on how many calls are added when a single atom t is added to $\mathcal{P}$. We say that a $\textsc{Lift}^{\mathrm{R}}_{\mathrm{O}}$-call *covers* an atom if that atom appears in its $\mathcal{P}$-argument. In Step 5, the separator variable must appear in every atom, which means that the separator variable must appear in t as well. Hence, of the child calls generated in Line 22, at most one can cover t. The number of calls that cover t is therefore bounded above by the number of recursive calls that can be generated in Steps 2–4. These steps are independent of $\mathcal{P}^r$, and only a function of the query. Therefore, the number of calls covering t is at most a constant in the size of $\mathcal{P}^r$. Every call to $\textsc{Lift}^{\mathrm{R}}_{\mathrm{O}}$ must cover at least one atom (ignoring the constant cost of Line 23 and its calls with empty databases), which bounds the number of calls to be linear in $\mathcal{P}^r$.

Second, we show that the computations inside each individual call to $\textsc{Lift}^{\mathrm{R}}_{\mathrm{O}}$ admit an overall linear complexity. When adding an atom t to $\mathcal{P}^r$, the calls that cover t are of two types: (1) calls that do not cover another atom in $\mathcal{P}^r$ except for t, and (2) calls that already cover another atom in $\mathcal{P}^r$.

(1) Because of database restriction operators such as $\mathcal{P}^r_{|x=c}$ throughout Algorithm 1, the calls of type 1 all have the minimum required database as an argument, that is, $|\mathcal{P}^r| = 1$. Thus we have a constant data complexity for non-recursive computations in calls of type 1.

(2) To analyze the complexity of type-2 calls, we make the standard assumption that the domain has a given one-to-one correspondence with the integers. This allows for the operation $\mathcal{P}^r_{|x=c}$ to be implemented in linear time for all c simultaneously (i.e., Step 5 has linear data complexity sans the recursive calls). The complexity of type-2 calls thus grows linearly with $\mathcal{P}^r$.

In total, adding an atom to $\mathcal{P}^r$ will add a constant number of type-1 calls, whose internal computations all have constant data complexity. It will also increase the runtime of a constant number of type-2 calls by a constant. This gives an overall linear data complexity.

Hence, the algorithm $\textsc{Lift}^{\mathrm{R}}_{\mathrm{O}}$ runs in time polynomial in data complexity (accounting also for preprocessing). The probability of any safe query can be evaluated in *linear time* in the size of the *ranked* OpenPDB $\mathcal{G}^r$ in data complexity. Moreover, any safe query which is self-join-free can be evaluated in *linear time* in the size of the input OpenPDB $\mathcal{G}$, since ranking is not needed for self-join free queries.

Proof of Theorem 6.4

For the membership results, it is sufficient to show that PQE(FO) is in PP in data complexity. Let $\mathcal{P}$ be a PDB, Q a FO query and $p \in (0, 1]$ a rational value. Let us denote by P the probability distribution induced by $\mathcal{P}$. We need to show that $\mathrm{P}(Q) > p$ can be decided in PP. Note that there are exponentially many databases (worlds) $\mathcal{D}$ induced by $\mathcal{P}$, each of which holds with some probability. We now create multiple copies of each world in such a way that the uniform distribution over all thus generated worlds is equivalent to P when each copy is taken to represent its original world. Given this uniform distribution over the worlds, we now consider a nondeterministic Turing machine, where each branch corresponds to one of these worlds. Each branch of the nondeterministic Turing machine represents an accepting run if the test $\mathcal{D} \models Q$ is positive for the corresponding world $\mathcal{D}$ (which can be verified in polynomial time in data complexity). Moreover, for threshold values properly above (respectively, below) 0.5, we introduce artificial success (respectively, failure) branches into the nondeterministic Turing machine such that satisfying the original threshold corresponds to having a majority of successful computations. Then, $\mathrm{P}(Q) > p$ (i.e., the answer to the probabilistic query entailment problem is *yes*) if and only if the nondeterministic Turing machine answers *yes* in the majority of its runs, which proves membership.

As for the hardness results, we first prove that PQE(∀FO) is PP-hard in data complexity. Let $\mathcal{P}$ be a PDB, Q a ∀FO query and $p \in (0, 1]$ a threshold value. We reduce from the following problem. Given a quantified Boolean formula $\Phi \coloneqq \mathsf{C}^c\, x_1, \ldots, x_n\ \phi$, where C represents the *counting quantifier* and $\phi = \phi_1 \wedge \cdots \wedge \phi_k$ is a propositional formula in 3CNF, defined over the variables $x_1, \ldots, x_n$, decide whether Φ is valid. Intuitively, this amounts to checking whether there are c assignments for $x_1, \ldots, x_n$ that satisfy ϕ, and deciding the validity of such formulas is PP-complete [33]. For the reduction, we consider the following ∀FO query:

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

which is used to encode the satisfaction conditions of the formula Φ. Furthermore, we define the PDB $\mathcal{P}_\Phi$ that stores the structure of Φ as follows.

- For each variable x_i, $1 \leq i \leq n$, $\mathcal{P}_\Phi$ contains the atoms $\langle \mathsf{L}(x_i) : 0.5 \rangle$, where we view each x_i as a database constant.
- The clauses ϕ_i are described with the help of the predicates R_1, …, R_4, each of which corresponds to one type of clause. More specifically, there are at most four different types of clauses in a 3CNF formula (modulo permutations): (i) R_1 encodes clauses with exactly three positive literals, (ii) R_2 encodes clauses with exactly two positive literals and a single negated literal, (iii) R_3 encodes clauses with exactly two negated literals and a single positive literal, (iv) R_4 encodes clauses with exactly three negated literals. For example, for the clause $\phi_i = x_1 \vee \neg x_2 \vee \neg x_4$, 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 $\neg\mathsf{L}(x_4)$, $\neg\mathsf{L}(x_2)$ or $\mathsf{L}(x_1)$ holds. All other R-atoms that do not correspond in such a way to one of the clauses, we add with probability 1 to $\mathcal{P}_\Phi$.

Claim. The formula Φ is valid if and only if $\mathrm{P}_{\mathcal{P}_\Phi}(Q_{\mathrm{SAT}}) \geq \mathsf{c} \cdot (0.5)^n$.

Suppose that Φ is valid. Then, there are at least c different assignments τ to the variables $x_1, \ldots, x_n$ that satisfy ϕ. For each satisfying assignment τ, there is a corresponding database $\mathcal{D}$ induced by $\mathcal{P}_\Phi$ such that (i) $\mathcal{D}$ contains an

atom $\mathsf{L}(x_i)$ if and only if τ sets x_i to true in Φ, and (ii) $\mathcal{D}$ contains all R-atoms that occur in $\mathcal{P}_\Phi$ with probability 1. For each such database $\mathcal{D}$, it holds that $\mathcal{D} \models Q_{\text{SAT}}$, as each such world is in one-to-one correspondence with a satisfying valuation. Note that there are only n probabilistic atoms in $\mathcal{P}_\Phi$, i.e., the atoms $\mathsf{L}(x_j)$, $1 \leq i \leq n$ (corresponding to the variables in Φ). Thus, every database $\mathcal{D}$ induced by $\mathcal{P}_\Phi$ has the probability 0.5^n. By our assumption, there are c satisfying assignments τ to Φ, and hence, it follows that $\text{P}_{\mathcal{P}_\Phi}(Q_{\text{SAT}}) \geq \mathsf{c} \cdot (0.5)^n$.

For the other direction, let $\text{P}_{\mathcal{P}_\Phi}(Q_{\text{SAT}}) \geq \mathsf{c} \cdot (0.5)^n$. Then, each database $\mathcal{D}$ induced by $\mathcal{P}_\Phi$ sets a choice for the nondeterministic atoms $\mathsf{L}(x_1), \ldots, \mathsf{L}(x_n)$ and each such database has the probability $(0.5)^n$ (as there are only n nondeterministic atoms in the PDB). As a consequence, there must exist at least c databases induced by $\mathcal{P}_\Phi$ that satisfies $\mathcal{D} \models Q$. For each such database $\mathcal{D}$, we define a corresponding assignment τ to the variables $x_1, \ldots, x_n$ such that x_i is mapped to true in τ if and only if $\mathsf{L}(x_i) \in \mathcal{D}$. It is then easy to verify that $\tau \models \phi$. As there are c different assignments τ that satisfy ϕ, we conclude that the formula Φ is valid. This proves PP-hardness for PQE(∀FO).

Observe that this hardness immediately applies to PQE(FO), as $Q_{\text{SAT}} \in \text{FO}$. Finally, note that the negation of Q_{SAT} is an ∃FO query and PP is closed under complement as it is closed under truth table reductions [30]. Hence, this hardness also holds for PQE(∃FO).

Proof of Theorem 6.5

We start by proving the membership results. It is sufficient to show that $\overline{\text{P}}\text{QE}(\text{FO}_\text{s})$ is in NP and $\underline{\text{P}}\text{QE}(\text{FO}_\text{s})$ is in coNP, since the membership results for the query classes $\exists\text{FO}_\text{s}$ and $\forall\text{FO}_\text{s}$ follow from these. Let $\mathcal{G} = (\mathcal{P}, \lambda)$ be an OpenPDB, Q be a FO_s query, and $p \in (0, 1]$ a rational value. We first show that deciding whether $\overline{\text{P}}_\mathcal{G}(Q) > p$ is in NP. To see why this holds, consider a nondeterministic Turing machine, where each computation branch corresponds to an extreme completion of $\mathcal{G}$. Each of these computation branches, corresponding to a completion $\hat{\mathcal{P}}$, can then be used to verify whether $\text{P}_{\hat{\mathcal{P}}}(Q) > p$. Note that this verification can be performed in polynomial time in data complexity, since any completion is a PDB, and the query $Q \in \text{FO}_\text{s}$ is safe for PDBs, by our assumption. Then, $\overline{\text{P}}_\mathcal{G}(Q) > p$ if and only if the described nondeterministic Turing machine answers yes. Hence, $\overline{\text{P}}\text{QE}(\text{FO}_\text{s})$ is in NP. To show that deciding whether $\underline{\text{P}}_\mathcal{G}(Q) > p$ is in coNP, we can prove that the complementary problem of deciding whether $\underline{\text{P}}_\mathcal{G}(Q) \leq p$ is in NP. To see why this holds, consider the same construction for a nondeterministic Turing machine as before, except that each computation branch now verifies whether $\text{P}_{\hat{\mathcal{P}}}(Q) \leq p$ (which can be done in polynomial time in data complexity), for the respective completion $\hat{\mathcal{P}}$ that it represents. Then, $\underline{\text{P}}_\mathcal{G}(Q) \leq p$ if and only if the described nondeterministic Turing machine answers yes. Hence, $\underline{\text{P}}\text{QE}(\text{FO}_\text{s})$ is in coNP.

For the hardness results, we prove $\overline{\text{P}}\text{QE}(\forall\text{FO}_\text{s})$ is NP-hard, which, by duality, implies $\underline{\text{P}}\text{QE}(\exists\text{FO}_\text{s})$ is coNP-hard. Clearly, these lower bounds apply to FO_s queries, and we obtain the all the claimed results. To prove this result, we carefully choose a query which is safe for PDBs, but becomes hard for OpenPDBs. More concretely, we carefully define a $\forall\text{FO}_\text{s}$ query Q_{SAFE} and prove the following propositions:

Proposition B.1. $\text{PQE}(Q_{\text{SAFE}})$ is in P for PDBs (i.e., Q_{SAFE} is *safe* for PDBs).

Proposition B.2. $\overline{\text{P}}\text{QE}(Q_{\text{SAFE}})$ is NP-hard for OpenPDBs.

Then, by Proposition B.1, it holds that $Q_{\text{SAFE}} \in \forall\text{FO}_\text{s}$, and together with Proposition B.2, we conclude that $\overline{\text{P}}\text{QE}(\forall\text{FO}_\text{s})$ is NP-hard for OpenPDB. The proof is thus structured as follows: (i) Identifying a query Q_{SAFE}, (ii) Proof of Proposition B.1, and (iii) Proof of Proposition B.2.

Identifying the query Q_{SAFE}

The construction of Q_{SAFE} is quite intricate. We start by describing this query, which, once identified, is fixed. The idea is to start from a ∀FO query:

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

which is shown to be *unsafe* for PDBs in the proof of Theorem 6.4. Observe also that the construction given in the proof of Theorem 6.4 shows that Q_{SAT} can encode an arbitrary propositional formula in 3CNF.

We apply a chain of transformations on Q_{SAT} in order to obtain Q_{SAFE}. First, we transform the query Q_{SAT} into a query Q_{EQ} that consists of individually *safe* clauses, while Q_{EQ} itself remains unsafe. Our transformation ensures that the queries Q_{SAT} and Q_{EQ} are equisatisfiable, and even more is actually true: there is a one-to-one mapping between the models of these queries. We then apply another transformation on the query to produce Q_{SAFE} from Q_{EQ}. Recall that Q_{EQ} is an unsafe query, but each of its clauses are safe. Q_{EQ} is hard since the terms produced in the inclusion-exclusion step are hard to evaluate. Put in more intuitive terms, clauses in Q_{EQ} are probabilistically dependent of each other, and this serves as the source for hardness. We manipulate these clauses so that they become mutually exclusive, which in turn helps us to come up with the Q_{SAFE}, that is safe for PDBs. Q_{SAFE} is defined in such a way that all the unsafe terms will cancel out during the exhaustive application of the inclusion-exclusion rule. This is the intuitive reason why Q_{SAFE} is safe for PDBs, while it is hard for OpenPDBs. We now provide the details of the corresponding transformations.

Transforming Q_{SAT} to Q_{EQ}. We show how to transform Q_{SAT} into an equisatisfiable query Q_{EQ}, using a special type of Tseitin transformation [91]. The idea is to detect the unsafe fragments in each clause of Q_{SAT} and replace them recursively with fresh atoms until the clause is safe. While doing so, we also add additional clauses to the formula, which assert the equivalence of the freshly introduced atom to the old formula, ensuring the overall equisatisfiability of Q_{SAT} and Q_{EQ}.

We omit the full details of this transformation (as it is well-known), but explain it on a small example. Consider, for instance a clause $\forall x, y\, \mathsf{L}(x) \vee \mathsf{L}(y) \vee \mathsf{R}_1(x, y)$, which is not safe as there is no separator variable. To transform this query, we define a fresh atom $\mathsf{Z}(x, y)$ to be equivalent to the formula $\mathsf{L}(y) \vee \mathsf{R}_1(x, y)$, which results in the following query:

$$\forall x, y\, (\mathsf{L}(x) \vee \mathsf{Z}(x, y)) \wedge (\mathsf{Z}(x, y) \leftrightarrow (\mathsf{L}(y) \vee \mathsf{R}_1(x, y))).$$

Notice that the first conjunct is already safe. The second conjunct can further be simplified as:

$$\begin{aligned}&\forall x, y\, (\ \ \mathsf{Z}(x, y) \rightarrow (\mathsf{L}(y) \vee \mathsf{R}_1(x, y))) \wedge ((\ \ \mathsf{L}(y) \vee \ \ \ \mathsf{R}_1(x, y)) \rightarrow \mathsf{Z}(x, y)) \equiv \\ &\forall x, y\, (\neg\mathsf{Z}(x, y) \vee \ \ \ \ \mathsf{L}(y) \vee \mathsf{R}_1(x, y)\ ) \wedge ((\neg\mathsf{L}(y) \wedge \neg\mathsf{R}_1(x, y)) \vee\ \mathsf{Z}(x, y)).\end{aligned}$$

Note that the first clause is also safe as there y is a separator variable (and afterwards x serves as a separator variable). However, the last clause is not in disjunctive form but can be decomposed into two disjunctive clauses:

$$(\neg\mathsf{L}(y) \vee \mathsf{Z}(x, y)) \wedge (\mathsf{Z}(x, y) \vee \neg\mathsf{R}_1(x, y)),$$

both of which are safe. Clearly, we can apply this transformation to any universally quantified formula, and it will eventually result in a (potentially) large conjunction of clauses, each of which is individually safe. As a consequence, we obtain a query $Q_{\text{EQ}} = \bigwedge q_i(x, y, z)$, where each $q_i(x, y, z)$ is a safe clause over the variables x, y and z. By construction, it is easy to see that any model of Q_{SAT} can be extended to a model of Q_{EQ}, by interpreting the freshly introduced atoms to be equivalent to the sub-formula they replaced.

Transforming Q_{EQ} to Q_{SAFE}. Every clause $q_i(x, y, z)$ in the query $Q_{\text{EQ}} = \bigwedge_{1\le i\le n} q_i(x, y, z)$ is safe, while the query itself is still not safe, and the reason is hidden in the inclusion-exclusion terms that are hard to evaluate. We define the following formula:

$$Q_{\text{SAFE}} = \ \forall x, y, z\ \bigwedge_{1\le i\le n} (q_i \vee \neg\mathsf{H}_i)\ \ \wedge \ \ \bigwedge_{1\le i<j\le n} (\neg\mathsf{H}_i \vee \neg\mathsf{H}_j)\ \ \wedge\ \ (\bigvee_{1\le i\le n} \mathsf{H}_i),$$

where H_i is a zero-arity predicate. Let us give some insight on this formula. Note that the second part of the formula consists of only H_i-atoms. The last clause simply says that at least one of the atoms H_i must be true. Together with the other clauses, the second part of the formula asserts that exactly one atom H_k, for some $1 \le k \le n$ can be true, and all the remaining atoms H_i, $1 \le i \ne k \le n$ must be false. This has direct implications on the first part of the formula where the clauses q_i from the original formula Q_{EQ} appear. Briefly stated, if we choose to satisfy the k-th atom, H_k, then this means all clauses $q_i \vee \neg\mathsf{H}_i$ will be trivially satisfied for $i \ne k$. Intuitively, this means that their influence on

the query probability will be fixed, and only $q_k \vee \neg \mathsf{H}_k$ will be counted. It is important to note that Q_{SAFE} and Q_{EQ} are *not* equisatisfiable.

We have now identified the query Q_{SAFE}, and based on this query, we prove the propositions.

Proposition B.1. PQE(Q_{SAFE}) is in P for PDBs (i.e., Q_{SAFE} is *safe* for PDBs).

Proof. Observe that the clauses in Q_{SAFE} of the form $\neg \mathsf{H}_i \vee q_i$ consist of multiple atoms that are not connected by a relational variable. That is, every clause consists of independent literals. Therefore, the entire query can be written as $(\neg \mathsf{H}_i \wedge \Delta) \vee (q_i \wedge \Delta)$, which is a UCNF. Applying this transformation to all clauses (all such clauses have disconnected H_i-atoms), we obtain a large union, on which we can perform inclusion-exclusion in Step 3. The detailed implementation of inclusion-exclusion for PDBs (cf. [52]) removes a large number of unsatisfiable CNF clauses from this union. Afterwards, all remaining CNF formulas in the union have the form:

$$\beta_k = \neg \mathsf{H}_1 \wedge \cdots \wedge \mathsf{H}_k \wedge q_k \wedge \ldots \neg \mathsf{H}_n,$$

that is, one H_i-atom for every i, and containing exactly one positive atom H_k with a corresponding clause $q_k(x, y, z)$. The entire UCNF is then given by $\bigvee_{i=1}^{n} \beta_i$. Importantly, the individual formulas β_i are mutually exclusive, removing the need for any nonsingular combination of β_i-terms in the inclusion-exclusion formula. Thus, the inclusion-exclusion rule computes:

$$\mathrm{P}_{\mathcal{P}}(Q_{\text{SAFE}}) = \sum_{i=1}^{m} \mathrm{P}_{\mathcal{P}}(\beta_i),$$

for some arbitrary PDB $\mathcal{P}$. Then, since q_i and H-atoms do not share any relation name, we can further decompose the query as:

$$\mathrm{P}_{\mathcal{P}}(\beta_i) = \mathrm{P}_{\mathcal{P}}(q_i) \cdot \mathrm{P}_{\mathcal{P}}(\neg \mathsf{H}_1 \wedge \cdots \wedge \mathsf{H}_i \wedge \ldots \neg \mathsf{H}_n),$$

where the first term of the multiplication is safe by construction and it is easy to see that the second term is also safe. Thus, we obtain:

$$\mathrm{P}_{\mathcal{P}}(Q_{\text{SAFE}}) = \sum_{i=1}^{m} \mathrm{P}_{\mathcal{P}}(q_i) \cdot \mathrm{P}_{\mathcal{P}}(\neg \mathsf{H}_1 \wedge \cdots \wedge \mathsf{H}_i \wedge \ldots \neg \mathsf{H}_n),$$

which allows us to conclude that Q_{SAFE} is safe for PDBs. □

Proposition B.2. $\overline{\text{P}}$QE(Q_{SAFE}) is NP-hard for OpenPDBs.

Proof. We give a reduction from the satisfiability problem defined over 3CNF formulas: given a propositional formula ϕ in a 3CNF, decide whether it is satisfiable. For our construction, we define an OpenPDB $\mathcal{G}_\phi = (\mathcal{P}_\phi, 1)$ over a vocabulary σ that contains the relation symbols from Q_{SAFE} (thus also from Q_{EQ}) as well as some additional constants (while assuming at least 3 of them). Intuitively, the fixed query Q_{SAFE} encodes (in a loose sense) the satisfaction conditions of the given 3CNF formula ϕ. The PDB $\mathcal{P}_\phi$ stores the structure of ϕ as follows:

- $\mathcal{P}_\phi$ contains all atoms $\langle \mathsf{H}_i : 0.5 \rangle$ for $1 \leq i \leq n$.
- The clauses ϕ_i are described with the help of the predicates $\mathsf{R}_1, \ldots, \mathsf{R}_4$, each of which corresponds to one type of clause, as described in the proof of Theorem 6.4. All other R-atoms that do not correspond in such a way to one of the clauses, we add with probability 1 to $\mathcal{P}_\phi$.
- All the remaining atoms (that are directly related to the satisfaction of Q_{EQ}) are left open. In other words, there are only n probabilistic atoms.

Claim. The formula ϕ in 3CNF is satisfiable if and only if $\overline{\mathrm{P}}_{\mathcal{G}_\phi}(Q_{\mathrm{SAFE}}) \geq n \cdot (0.5)^n$.

Let us assume that $\overline{\mathrm{P}}_{\mathcal{G}_\phi}(Q_{\mathrm{SAFE}}) \geq n \cdot (0.5)^n$. This implies that there exists a completion that sets a choice for the open atoms such that the probability of Q_{SAFE} relative to this completion is at least $n \cdot (0.5)^n$. By the structure of Q_{SAFE}, we already know that there are n different configurations of the H_i-atoms, each with probability $(0.5)^n$. This means that all $q_i(x, y, z)$, $1 \leq i \leq n$ must be satisfied by a distinct database with probability $(0.5)^n$. Note, however, all these databases differ only with respect to the H_i-atoms. Apart from H-atoms, all the atoms are deterministic in the completion; that is, they either are in the completion with probability 1, or are excluded. This implies that there exists a database $\mathcal{D}$ that satisfies all $q_i(x, y, z)$, $1 \leq i \leq n$. We now define a propositional assignment τ such that it maps a variable u in ϕ to true if and only if $\mathsf{L}(u) \in \mathcal{D}$. It is then easy to show that $\tau \models \phi$.

Conversely, let us assume that ϕ is satisfiable and let τ be a satisfying assignment of ϕ. We define the completion $\mathcal{P}_\tau$ as follows. First note that the completion $\mathcal{P}_\tau \supseteq \mathcal{P}_\Phi$, i.e., all atoms $\langle \mathsf{H}_i : 0.5 \rangle$ for $1 \leq i \leq n$ are in the completion, and so are the R_i atoms from $\mathcal{P}_\Phi$. Moreover, we add $\langle \mathsf{L}(u) : 1 \rangle$ if τ maps u to true; otherwise, we add $\langle \mathsf{L}(u) : 0 \rangle$ to the completion $\mathcal{P}_\tau$. Notice that all Z-atoms are introduced to be equivalent to some L-atom (in the construction of the query). To preserve this, we also add the respective Z-atoms, either with probability 0, or 1, depending on the L-atoms.

As before, there are n configurations of H_i-atoms, each with 0.5^n probability. For each such configuration exactly one q_i must be satisfied, which holds since each database $\mathcal{D}$ induced by $\mathcal{P}_\tau$ differs only on the H-atoms and it is easy to verify that each of them satisfies $\mathcal{D} \models q_i$ for all $1 \leq i \leq n$. Thus, we obtain that $\overline{\mathrm{P}}_{\mathcal{G}_\phi}(Q_{\mathrm{SAFE}}) \geq \mathrm{P}_{\mathcal{P}_\tau}(Q_{\mathrm{SAFE}}) = n \cdot (0.5)^n$, which concludes the proof of Proposition B.2. □

We have thus proven all membership and hardness results stated in Theorem 6.5.

Proof of Theorem 6.6

We start by proving the membership results. As before, it is sufficient to show that $\overline{\mathrm{P}}$QE(FO) is in $\mathrm{NP}^{\mathrm{PP}}$ and $\underline{\mathrm{P}}$QE(FO) is in $\mathrm{coNP}^{\mathrm{PP}}$, and these cover all membership results for all query classes under consideration. Let $\mathcal{G} = (\mathcal{P}, \lambda)$ be an OpenPDB, Q be a FO query, and $p \in (0, 1]$ a rational value. To show that deciding whether $\overline{\mathrm{P}}_{\mathcal{G}}(Q) > p$ is in $\mathrm{NP}^{\mathrm{PP}}$, consider a nondeterministic Turing machine with a PP oracle such that each computation branch corresponds to one of the extreme completions $\hat{\mathcal{P}}$ of $\mathcal{G}$ (as in the proof of Theorem 6.5), and, for each such branch, the corresponding verification $\mathrm{P}_{\hat{\mathcal{P}}}(Q) > p$ is done by the PP oracle. Note that the verification $\mathrm{P}_{\hat{\mathcal{P}}}(Q) > p$ can be performed in PP in data complexity, since the completion $\mathrm{P}_{\hat{\mathcal{P}}}$ is a PDB, and PQE(FO) is in PP for PDBs. Then, $\overline{\mathrm{P}}_{\mathcal{G}}(Q) > p$ if and only if the described nondeterministic Turing machine answers yes. Hence, $\overline{\mathrm{P}}$QE(FO) is in $\mathrm{NP}^{\mathrm{PP}}$. To show that deciding whether $\underline{\mathrm{P}}_{\mathcal{G}}(Q) > p$ is in $\mathrm{coNP}^{\mathrm{PP}}$, we prove that the complementary problem of deciding whether $\underline{\mathrm{P}}_{\mathcal{G}}(Q) \leq p$ is in $\mathrm{NP}^{\mathrm{PP}}$. Consider the same construction for the nondeterministic Turing machine as before, except that each computation branch uses the oracle, for verifying $\mathrm{P}_{\hat{\mathcal{P}}}(Q) \leq p$. This verification is in PP, since it is the complement of deciding $\mathrm{P}_{\hat{\mathcal{P}}}(Q) > p$, which is PP-complete, and PP is closed under complement. Then, $\underline{\mathrm{P}}_{\mathcal{G}}(Q) \leq p$ if and only if the described nondeterministic Turing machine answers yes.

For the hardness results, we prove (i) $\overline{\mathrm{P}}$QE($\forall$FO) is $\mathrm{NP}^{\mathrm{PP}}$-hard for OpenPDBs, and (ii) $\underline{\mathrm{P}}$QE($\forall$FO) is $\mathrm{coNP}^{\mathrm{PP}}$-hard for OpenPDBs. By duality, (i) and (ii) imply the results for $\overline{\mathrm{P}}$QE($\exists$FO), and $\underline{\mathrm{P}}$QE($\exists$FO). Clearly, all these lower bounds apply to FO queries, and we obtain all the claimed results.

Proposition B.3. $\overline{\mathrm{P}}$QE($\forall$FO) is $\mathrm{NP}^{\mathrm{PP}}$-hard for OpenPDBs.

Proof. We reduce from the following problem. Let $\Phi = \exists x_1, \ldots, x_\ell\, \mathsf{C}^c\, y_1, \ldots, y_m\, \phi$, denote a quantified Boolean formula, where C represents the *counting quantifier* and $\phi = \phi_1 \wedge \cdots \wedge \phi_k$ is a propositional formula in 3CNF, defined over the variables $x_1, \ldots, x_\ell, y_1, \ldots, y_m$. Deciding the validity of such formulas is $\mathrm{NP}^{\mathrm{PP}}$-complete [33]. Intuitively, this amounts to checking whether there is a partial assignment for $x_1, \ldots, x_\ell$ that admits at least c extensions to $y_1, \ldots, y_m$ that satisfy ϕ.

To reduce the problem to upper probabilistic query evaluation, we consider again the universally quantified query Q_{SAT} given in the proof of Theorem 6.5. As before, Q_{SAT} is used to encode the satisfaction conditions of the formula Φ together with the PDB $\mathcal{P}_\Phi$ that stores the structure of Φ. The PDB $\mathcal{P}_\Phi$ is given as follows: for each variable y_j, $1 \leq j \leq m$, $\mathcal{P}_\Phi$ contains the atoms $\langle \mathsf{L}(y_j) : 0.5 \rangle$, where we view each y_j as a constant. As in the proof of Theorem 6.5, the clauses ϕ_i are described with the help of the predicates $\mathsf{R}_1, \ldots, \mathsf{R}_4$. All other R-atoms that do not

correspond in such a way to one of the clauses, we add with probability 1 to $\mathcal{P}_\Phi$. Notice that the atoms $\langle \mathsf{L}(x_i) : 0.5\rangle$, $1 \leq i \leq l$ that correspond to the x-variables in Φ are left open. Finally, we define the OpenPDB $\mathcal{G}_\Phi = (\mathcal{P}_\Phi, 1)$. The construction provided for Q_{SAT} and $\mathcal{G}_\Phi$ is clearly polynomial. Furthermore, the query is fixed, and only $\mathcal{P}_\Phi$ depends on Φ. We now prove the following claim.

Claim. The formula Φ is valid if and only if $\overline{\mathrm{P}}_{\mathcal{G}_\Phi}(Q_{\text{SAT}}) \geq \mathsf{c} \cdot (0.5)^m$.

Suppose that Φ is valid. Then, for some assignment μ of the variables $x_1, \ldots, x_\ell$, there are at least c different assignments τ extending μ to the variables $y_1, \ldots, y_m$ that satisfy Φ. We use the assignment μ in order to set a choice for all open atoms $\mathsf{L}(x_i)$, $1 \leq i \leq \ell$. More precisely, we define the λ-completion $\mathcal{P}_\mu$ that contains $\langle \mathsf{L}(x_i) : 1\rangle$ if μ sets x_i to true and contains $\langle \mathsf{L}(x_i) : 0\rangle$, otherwise. Intuitively, every assignment of the existentially quantified variables in Φ corresponds to a different completion and the assignment μ is realized by the completion $\mathcal{P}_\mu$.

Moreover, observe that for each satisfying assignment τ extending μ to the variables $y_1, \ldots, y_m$, there exists a database $\mathcal{D}$ induced by $\mathcal{P}_\mu$. We can define such a database $\mathcal{D}$ as follows: add all atoms to $\mathcal{D}$ that are in $\mathcal{P}_\mu$ with probability 1 and add every atom $\mathsf{L}(y_j)$ to $\mathcal{D}$ if and only if τ sets y_i to true. It is easy to see that each such database satisfies $\mathcal{D} \models Q_{\text{SAT}}$. Finally, it suffices to observe that there are only m nondeterministic atoms in $\mathcal{P}_\mu$; namely the atoms $\mathsf{L}(y_j)$, $1 \leq j \leq m$ that correspond to the y-variables in Φ. Thus, every database $\mathcal{D}$ induced by $\mathcal{P}_\mu$ has the probability 0.5^m. By our assumption, there are c satisfying assignments τ extending μ; thus, it follows that $\mathrm{P}_{\mathcal{P}_\mu}(Q_{\text{SAT}}) = \mathsf{c} \cdot (0.5)^m$, which implies $\overline{\mathrm{P}}_{\mathcal{G}_\Phi}(Q_{\text{SAT}}) \geq \mathsf{c} \cdot (0.5)^m$ as a consequence of the query semantics in OpenPDBs.

For the other direction, let $\overline{\mathrm{P}}_{\mathcal{G}_\Phi}(Q_{\text{SAT}}) \geq \mathsf{c} \cdot (0.5)^m$. Then, there exists a λ-completion $\mathcal{P}_\mu$ such that $\mathrm{P}_{\mathcal{P}_\mu}(Q_{\text{SAT}}) \geq \mathsf{c} \cdot (0.5)^m$. Moreover, each database $\mathcal{D}$ induced by $\mathcal{P}_\mu$ sets a choice for the nondeterministic atoms $\mathsf{L}(y_1), \ldots, \mathsf{L}(y_m)$ and each such database has the probability $(0.5)^m$ (as there are only m nondeterministic atoms in the PDB). As a consequence, there must exist at least c databases induced by $\mathcal{P}_\mu$ that satisfies $\mathcal{D} \models Q$.

We define an assignment μ to the variables $x_1, \ldots, x_\ell$ such that x_i is mapped to true in μ if and only if $\langle \mathsf{L}(x_i) : 1\rangle \in \mathcal{P}_\mu$. Then, for each database $\mathcal{D}$ induced by $\mathcal{P}_\mu$ and that satisfies $\mathcal{D} \models Q_{\text{SAT}}$, we define an assignment τ that sets y_j to true if and only if $\mathsf{L}(y_j) \in \mathcal{D}$. It is then easy to verify that $\tau \models \Phi$ and that τ properly extends μ to a complete assignment. As there are c different assignments τ that extend μ while satisfying ϕ, we conclude that the formula Φ is valid. □

Proposition B.4. $\underline{\text{P}}$QE($\forall$FO) is coNP$^{\text{PP}}$-hard for OpenPDBs.

Proof. This proof follows similar ideas to the proof of Proposition B.3. We reduce from the problem of deciding validity of formulas of the following form $\Phi = \forall x_1, \ldots, x_\ell\, \mathsf{C}^c\, y_1, \ldots, y_m\, \phi$, which is similar to the earlier problem, except that the x-variables are now universally quantified. We use the exact same construction for $\mathcal{G} = (\mathcal{P}_\Phi, 1)$ and Q_{SAT} as before with the only difference being that the x-variables are now universally quantified.

Claim. The formula Φ is valid if and only if $\underline{\mathrm{P}}_{\mathcal{G}_\Phi}(Q_{\text{SAT}}) \geq \mathsf{c} \cdot (0.5)^m$.

Suppose that Φ is valid. Consider any completion $\mathcal{P}_\lambda$ that sets a choice for the open atoms $\mathsf{L}(x_1), \ldots, \mathsf{L}(x_\ell)$. We define an instantiation μ such that μ maps x_i to true if and only if $\langle \mathsf{L}(x_i) : 1\rangle$ is in the PDB $\mathcal{P}_\lambda$. Since Φ is valid, we know that for *any* instantiation of the variables $x_1, \ldots, x_\ell$, there exists at least c assignments τ that extends this instantiation to the variables $y_1, \ldots, y_m$ satisfying ϕ. Thus, there must exists at least c assignments τ extending μ such that $\tau \models \phi$.

It is easy to see that each such assignment τ defines a database $\mathcal{D}_\tau$ induced by the PDB $\mathcal{P}_\lambda$ and that $\mathcal{D}_\tau \models Q_{\text{SAT}}$. As before, every $\mathcal{D}_\tau$ has the probability 0.5^m since there are m nondeterministic atoms. This proves that, for any completion, the probability of the given query cannot be less than $\mathsf{c} \cdot (0.5)^m$, which yields $\underline{\mathrm{P}}_{\mathcal{G}_\Phi}(Q_{\text{SAT}}) \geq \mathsf{c} \cdot (0.5)^m$.

For the other direction, we know that regardless of the completion $\mathcal{P}_\lambda$ that is chosen, it holds that $\mathrm{P}_\lambda(Q_{\text{SAT}}) \geq \mathsf{c} \cdot (0.5)^m$. Moreover, every completion corresponds to a valuation μ of the x-variables in Φ and for each such assignment can be extended to c satisfying assignments, as before.

Observe that every world induced by a completion $\mathcal{P}_\lambda$ has the probability 0.5^m. To satisfy $\mathrm{P}_{\mathcal{G}_\lambda}(Q_{\text{SAT}}) \geq \mathsf{c} \cdot (0.5)^m$, there have to be c databases induced by $\mathcal{P}_\lambda$ satisfying Q_{SAT}. We have shown that each such database $\mathcal{D}_\tau$ corresponds to a satisfying assignment τ that extends μ such that $\tau \models \phi$. Hence, there must be at least c such assignments. Finally, since we proved this for an arbitrary completion (hence, for an arbitrary valuation of the x-variables), we conclude that the Φ is valid. □

We have thus proven all membership and hardness results stated in Theorem 6.6.

Appendix C. Proofs of Combined Complexity Results

This part contains all the proofs of the combined complexity results stated in Section 7.

Proof of Theorem 7.1

We first show that PQE(∃FO) is in PP^{NP} in combined complexity. Let $\mathcal{P}$ be a PDB, Q a ∃FO query and $p \in (0, 1]$ a threshold value. To decide whether $\text{P}(Q) > p$, consider a nondeterministic Turing machine as described in the proof of Theorem 6.4, but one, which additionally has access to an NP oracle. The only difference is that the verification step $\mathcal{D} \models Q$, for each world $\mathcal{D}$ induced by the PDB $\mathcal{P}$, is NP-complete in combined complexity. The answer to this test can be retrieved from the oracle machine. Then, $\text{P}(Q) > p$ (i.e., the answer to the probabilistic query entailment problem is *yes*) if and only if the nondeterministic Turing machine answers *yes* in the majority of its runs. This proves that PQE(∃FO) (and hence PQE(UCQ)) is in PP^{NP} in combined complexity. Finally, observe that PQE(∀FO) is also in PP^{NP}, since query evaluation for ∀FO queries is coNP-complete, i.e., the same oracle can be called for the verification step $\mathcal{D} \models Q$ for universal queries.

In order to show hardness, we reduce from the following problem: decide validity of formulas of the form $\Phi = \mathsf{C}^c\, x_1, \ldots, x_m \exists y_1, \ldots, y_n\, \phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_k$, where every ϕ_i is a propositional clause over $x_1, \ldots, x_m$, $y_1, \ldots, y_n$, and $k, m, n \geq 1$. Φ is valid if and only if, for at least c of the partial assignments μ to $x_1, \ldots, x_m$, the formula $\exists y_1, \ldots, y_n\, \mu(\phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_k)$ is true. This is a PP^{NP}-complete problem [33]. To simplify the proof, we also assume, without loss of generality, that ϕ contains all clauses of the form $x_j \vee \neg x_j$, $1 \leq j \leq m$, and similarly $y_j \vee \neg y_j$, $1 \leq j \leq n$; clearly, this does not affect the existence or number of satisfying assignments for ϕ. We also assume that each clause ϕ_j contains exactly three literals. This is also without loss of generality, since otherwise we can introduce additional existentially quantified variables to abbreviate the clauses, or duplicate literals if the clauses are too short.

We can now describe the construction for a PDB and a query. We define the PDB $\mathcal{P}_\Phi$ for the reduction as follows:

- For each variable x_j, $1 \leq j \leq m$, $\mathcal{P}_\Phi$ contains the atoms $\langle \mathsf{L}(x_j, 0) : 0.5 \rangle$ and $\langle \mathsf{L}(x_j, 1) : 0.5 \rangle$.
- Each clause ϕ_j is described with the help of a predicate $\mathsf{M}(\cdot, \cdot, \cdot, j)$ of arity 4, which encodes the satisfying assignments for ϕ_j. For example, consider the clause $\phi_j = x_2 \vee \neg y_4 \vee y_1$. For the satisfying assignment $x_2 \mapsto \mathit{true}, y_4 \mapsto \mathit{true}, y_1 \mapsto \mathit{false}$, we add the atom $\mathsf{M}(1, 1, 0, j)$ with probability 1, and similarly for all other satisfying assignments. There are at most 7 satisfying assignments for each clause.

Furthermore, we define the UCQ:

$$Q_\Phi = (\exists y_1, \ldots, y_n\, \psi_1 \wedge \cdots \wedge \psi_k) \vee (\exists x\, \mathsf{L}(x, 0) \wedge \mathsf{L}(x, 1)),$$

where each ψ_j is a conjunction that is derived from ϕ_j depending on the types of the involved variables. We describe the details again on the example clause $\phi_j = x_2 \vee \neg y_4 \vee y_1$. The satisfaction of this clause is encoded by the conjunction

$$\psi_j = \mathsf{M}(i, y_4, y_1, j) \wedge \mathsf{L}(x_2, i),$$

where i is an additional existentially quantified variable that is local to ψ_j, and j is fixed. Intuitively, ψ_j asserts that the truth assignment for x_2, y_4, and y_1 (given by x_2, i, and y_1, respectively) satisfies ϕ_j. Note that the variables $y_1, \ldots, y_n$ have to be mapped to 0 or 1, since otherwise they cannot satisfy the M-atoms. Moreover, observe that an alternative way of satisfying Q_Φ is due to the last clause in Q_Φ: it applies when L-atoms represent an inconsistent assignment (in Φ) for at least one variable of the form x_j. Note that, in this case, the query can be satisfied without actually satisfying the original formula Φ. This happens only if the the world contains both $\mathsf{L}(x_j, 0)$, $\mathsf{L}(x_j, 1)$. As there are $2m$ nondeterministic atoms in $\mathcal{P}_\Phi$, there are 4^m worlds; among them, $(4^m - 3^m)$ satisfy the last clause of the query Q_Φ, which corresponds to an inconsistent valuation in Φ. Note that there are other inconsistent assignments, namely, those that exclude both $\mathsf{L}(x_j, 0)$ and $\mathsf{L}(x_j, 1)$, but these cannot satisfy the query. Based on the given construction, and these observations, we now prove the following claim.

Claim. Φ is valid if and only if $\text{P}_{\mathcal{P}_\Phi}(Q_\Phi) \geq 0.5^{2m}(4^m - 3^m + \mathsf{c})$.

Suppose that Φ is valid. Then, there are at least c assignments μ for $x_1, \ldots, x_m$ such that each of these assignments admit an extension τ to the variables $y_1, \ldots, y_n$ such that $\tau \models \phi$. For each partial valuation μ, we define a database $\mathcal{D}_\mu$ such that it contains all atoms from $\mathcal{P}_\Phi$ that occur with probability 1. Moreover, $\mathcal{D}_\mu$ 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 $\mathcal{D}_\mu$ is induced by the PDB $\mathcal{P}_\Phi$. Besides, since each of these assignments μ admit an extension τ to the variables $y_1, \ldots, y_n$ such that $\tau \models \phi$, it follows that $\mathcal{D}_\mu \models (\exists y_1, \ldots, y_n\, \psi_1 \wedge \cdots \wedge \psi_k)$, as all satisfying assignments are already encoded in the database. In particular, this implies that $\mathcal{D}_\mu \models Q_\Phi$ for c worlds. Recall also that $(4^m - 3^m)$ worlds satisfy the last clause in the query (which captures the inconsistent valuations). As every world has the probability $(0.5)^{2m}$, we conclude that $\mathrm{P}_{\mathcal{P}_\Phi}(Q_\Phi) \geq 0.5^{2m}(4^m - 3^m + c)$.

Conversely, if $\mathrm{P}_{\mathcal{P}_\Phi}(Q_\Phi) \geq 0.5^{2m}(4^m - 3^m + c)$, then there are at least c worlds that satisfy the first clause in Q_Φ. For each of those worlds $\mathcal{D}$, we define a partial assignment $\mu_\mathcal{D}$ such that a variable x_j is mapped to true if $\mathsf{L}(x_j, 1) \in \mathcal{D}$ and it is mapped to false if $\mathsf{L}(x_j, 0)$. Moreover, $\mathcal{D} \models (\exists y_1, \ldots, y_n\, \psi_1 \wedge \cdots \wedge \psi_k)$ implies that there is a satisfying mapping for the y-variables in the database. Recall that, this can only be the case if a variable y_j is either mapped to 0 or to 1 due to the structure encoded in M-atoms. We define an extension $\tau_\mathcal{D}$ of $\mu_\mathcal{D}$, which maps a variable y_j to true if and only if it is mapped to 1 in the database and to false, otherwise. It is easy to verify that $\tau_\mathcal{D} \models \phi$. Thus, for c partial assignments, the formula $\exists y_1, \ldots, y_n\, \phi_1 \wedge \cdots \wedge \phi_k$ is satisfiable; meaning that, the formula Φ must be valid.

Proof of Theorem 7.2

We first show that PQE(FO) is in PSpace in combined complexity. Let $\mathcal{P}$ be a PDB, Q a FO query and $p \in (0, 1]$ a threshold value. To decide whether $\mathrm{P}_\mathcal{P}(Q) > p$, consider a *polynomial-space bounded* nondeterministic Turing machine that enumerates all (exponentially many) worlds $\mathcal{D}$, while keeping one world in memory at a time, and performs the test $\mathcal{D} \models Q$ for those worlds; then, adds up their probabilities if the test is successful. Note that the test $\mathcal{D} \models Q$ can be performed in polynomial, since the query evaluation problem for FO queries is PSpace-complete. Finally, the machine answers *yes* if and only if $\mathrm{P}_\mathcal{P}(Q) > p$, which proves membership.

Hardness is an immediate consequence of the fact that probabilistic query evaluation is a generalization of query evaluation, and query evaluation for FO queries is PSpace-hard in combined complexity (even if we assume that the arity of the predicates are bounded). Specifically, consider an arbitrary database $\mathcal{D}$ and a FO query Q. To decide whether $\mathcal{D} \models Q$, we define a PDB $\mathcal{P}$, which contains all the atoms from $\mathcal{D}$ with probability 1. Then, $\mathcal{D} \models Q$ if and only if $\mathrm{P}_\mathcal{P}(Q) \geq 1$.

Proof of Theorem 7.3

For the membership results, it is sufficient to show that $\overline{\mathrm{P}}$QE(UCQ) is in $\mathrm{PP}^{\mathrm{NP}}$ (since $\underline{\mathrm{P}}$QE(UCQ) coincides with $\underline{\mathrm{P}}$QE(UCQ)). Let $\mathcal{G} = (\mathcal{P}, \lambda)$ be an OpenPDB, Q be a UCQ, and $p \in (0, 1]$ a rational value. To decide whether $\overline{\mathrm{P}}_\mathcal{G}(Q) > p$, we consider the completion $\mathcal{P}_\lambda$ of $\mathcal{P}$, which sets the probability of all open atoms to λ. By Theorem 5.9 and by the monotonicity of UCQs, this completion maximizes the query probability, and since the arity of the predicates is bounded, the size of this completion is also bounded by a polynomial. Thus, for a UCQ Q, we have reduced $\overline{\mathrm{P}}_\mathcal{G}(Q) > p$ to $\mathrm{P}_{\mathcal{P}_\lambda}(Q) > p$, which is in $\mathrm{PP}^{\mathrm{NP}}$ by Theorem 7.1. Hardness also follows from Theorem 7.1, which asserts that PQE(UCQ) is $\mathrm{PP}^{\mathrm{NP}}$-hard for PDBs.

Proof of Theorem 7.4

We prove the results for $\overline{\mathrm{P}}$QE($\exists$FO), and $\underline{\mathrm{P}}$QE($\exists$FO) in bounded-arity combined complexity, and by the duality property, the results for universal queries are implied. Let $\mathcal{G} = (\mathcal{P}, \lambda)$ be an OpenPDB, Q a $\exists$FO query and $p \in [0, 1)$. To decide whether $\overline{\mathrm{P}}_\mathcal{G}(Q) > p$, consider a nondeterministic Turing machine with a $\mathrm{PP}^{\mathrm{NP}}$ oracle. The nondeteterministic Turing machine is used to guess a completion $\hat{\mathcal{P}}$ (that is of size polynomial in bounded-arity complexity) and then for verifying whether $\mathrm{P}_{\hat{\mathcal{P}}}(Q) > p$. Since $\mathrm{P}_{\hat{\mathcal{P}}}$ is a PDB, this verification can be done using the $\mathrm{PP}^{\mathrm{NP}}$ oracle as shown in Theorem 7.1. This implies an upper bound $\mathrm{NP}^{\mathfrak{C}}$, where $\mathfrak{C} = \mathrm{PP}^{\mathrm{NP}}$. Then, using Toda's result [32], which asserts that $\mathrm{PP}^{\mathrm{PH}} \subseteq \mathrm{P}^{\mathrm{PP}}$, it is easy to see that $\mathfrak{C} \subseteq \mathrm{P}^{\mathrm{PP}}$ and thus $\mathrm{NP}^{\mathfrak{C}} = \mathrm{NP}^{\mathrm{PP}}$. To show that deciding whether $\underline{\mathrm{P}}_\mathcal{G}(Q) > p$ in bounded-arity combined complexity is in $\mathrm{coNP}^{\mathrm{PP}}$, we can show that the complement problem, i.e., deciding $\underline{\mathrm{P}}_\mathcal{G}(Q) \leq p$ in bounded-arity combined complexity is in $\mathrm{NP}^{\mathrm{PP}}$. This can be shown using the same construction, except that the verification step checks for $\mathrm{P}_{\hat{\mathcal{P}}}(Q) \leq p$, after identifying the right completion. This verification can also be done using the $\mathrm{PP}^{\mathrm{NP}}$ oracle since the complement of this check is in $\mathrm{PP}^{\mathrm{NP}}$ by Theorem 7.1, and $\mathrm{PP}^{\mathrm{NP}}$ is closed under complement.

This concludes all membership results in bounded-arity combined complexity, and all hardness results in bounded-arity combined complexity follow from the hardness results given for data complexity, i.e., by Theorem 6.6.

Proof of Theorem 7.5

We prove $\overline{\text{P}}$QE(FO) and $\underline{\text{P}}$QE(FO) are PSpace-complete in bounded-arity combined complexity. Let $\mathcal{G} = (\mathcal{P}, \lambda)$ be an OpenPDB, Q a FO query and $p \in [0, 1)$. To decide whether $\overline{\text{P}}_{\mathcal{G}}(Q) > p$, consider a *polynomial space bounded* nondeterministic Turing machine that enumerates (exponentially many) extreme completions, each of which is of polynomial size, keeping only one such completion in memory at a time. Then, for each of these completions $\hat{\mathcal{P}}$, it tests whether $\text{P}_{\hat{\mathcal{P}}}(Q) > p$, which is in PSpace by Theorem 7.2. By similar ideas, we can decide whether $\underline{\text{P}}_{\mathcal{G}}(Q) > p$ using a polynomial space bounded nondeterministic Turing machine. PSpace-hardness follows from the hardness given for probabilistic query evaluation given in Theorem 7.2.