Foundations and Trends® in Databases
Vol. 7, No. 3-4 (2015) 197–341

DOI: 10.1561/1900000052

now
the essence of knowledge

Query Processing on Probabilistic Data: A Survey

Guy Van den Broeck
University of California
Los Angeles

Dan Suciu
University of Washington
Seattle

Contents

Abstract

Probabilistic data is motivated by the need to model uncertainty in large databases. Over the last twenty years or so, both the Database community and the AI community have studied various aspects of probabilistic relational data. This survey presents the main approaches developed in the literature, reconciling concepts developed in parallel by the two research communities. The survey starts with an extensive discussion of the main probabilistic data models and their relationships, followed by a brief overview of model counting and its relationship to probabilistic data. After that, the survey discusses lifted probabilistic inference, which are a suite of techniques developed in parallel by the Database and AI communities for probabilistic query evaluation. Then, it gives a short summary of query compilation, presenting some theoretical results highlighting limitations of various query evaluation techniques on probabilistic data. The survey ends with a very brief discussion of some popular probabilistic data sets, systems, and applications that build on this technology.

G. Van den Broeck and D. Suciu. *Query Processing on Probabilistic Data: A Survey*. Foundations and Trends® in Databases, vol. 7, no. 3-4, pp. 197–341, 2015.
DOI: 10.1561/1900000052.

1

Introduction

The goal of probabilistic databases is to manage large volumes of data that is uncertain, and where the uncertainty is defined by a probability space. The idea of adding probabilities to relational databases goes back to the early days of relational databases [Gelenbe and Hébrail, 1986, Cavallo and Pittarelli, 1987, Barbará et al., 1992], motivated by the need to represent NULL or unknown values for certain data items, data entry mistakes, measurement errors in data, "don't care" values, or summary information [Gelenbe and Hébrail, 1986]. Today, the need to manage uncertainty in large databases is even more pressing, as structured data is often acquired automatically by extraction, integration, or inference from other large data sources. The best known examples of large-scale probabilistic datasets are probabilistic knowledge bases such as Yago [Hoffart et al., 2013], Nell [Carlson et al., 2010], DeepDive [Shin et al., 2015], Reverb [Fader et al., 2011], Microsoft's Probase [Wu et al., 2012] or Google's Knowledge Vault [Dong et al., 2014], which have millions to billions of uncertain tuples.

Query processing in databases systems is a mature field. Techniques, tradeoffs, and complexities for query evaluation on all possible hardware architectures have been studied intensively [Graefe,

1993, Chaudhuri, 1998, Kossmann, 2000, Abadi et al., 2013, Ngo et al., 2013], and many commercial or open-source query engines exists today that implement these algorithms. However, query processing on *probabilistic data* is quite a different problem, since now, in addition to traditional data processing we also need to do probabilistic inference. A typical query consists of joins, projections with duplicate removal, grouping and aggregation, and/or negation. When the input data is probabilistic, each tuple in the query answer must be annotated with a probability: computing this output probability is called *probabilistic inference*, and is, in general, a challenging problem. For some simple queries, probabilistic inference is very easy: for example, when we join two input relations, we can simply multiply the probabilities of the tuples from the two inputs, assuming they are independent events. This straightforward approach was already used by Barbará et al. [1992]. But for more complex queries probabilistic inference is challenging.

The query evaluation problem over probabilistic databases has been studied over the last twenty years [Fuhr and Rölleke, 1997, Lakshmanan et al., 1997, Dalvi and Suciu, 2004, Benjelloun et al., 2006a, Antova et al., 2007, Olteanu et al., 2009]. In general, query evaluation, or, better, the probabilistic inference sub-problem of query evaluation, is equivalent to weighted model counting on a Boolean formula, a problem well studied in the theory community, as well as the AI and automated reasoning communities. While weighted model counting is known to be #P-hard in general, it has been shown that, for certain queries, probabilistic inference can be done efficiently. Even better, such a query can be rewritten into a (more complex) query, which computes probabilities directly using simple operations (sum, product, and difference). Therefore, query evaluation, including probabilistic inference, can be done entirely in one of today's relational database engines. Such a query can benefit immediately from decades of advances in query processing, including indexes, query optimization, parallel processing, etc. However, for other queries, computing their output probability is #P-hard. In this case the probabilistic inference task far dominates the query evaluation cost, and these hard queries are typically evaluated using some approximate methods for

weighted model counting. Dalvi and Suciu [2004] proved that, for a simple class of queries, either the query can be computed in polynomial time in the size of the database, by pushing the probabilistic inference in the engine, or the query's output probability is provably #P-hard to compute. Thus, we have a dichotomy: every query is either efficiently computable, or provably hard, and the distinction can be made using static analysis on the query.

Probabilistic graphical models preceded probabilistic databases, and were popularized in a very influential book by Pearl [1988]. In that setting, the knowledge base is described by a graph, such as a Bayesian or Markov network. Probabilistic inference on graphical models is also #P-hard in the size of the graph. Soon the AI community noticed that this graph often results from a concise relational representation [Horsch and Poole, 1990, Poole, 1993, Jaeger, 1997, Ngo and Haddawy, 1997, Getoor and Taskar, 2007]. Usually the relational representation is much more compact than the resulting graph, raising the natural question whether probabilistic inference can be performed more efficiently by reasoning on the relational representation instead of the grounded graphical model. This lead to the notion of *lifted inference* [Poole, 2003], whose goal is to perform inference on the high-level relational representation without having to ground the model. Lifted inference techniques in AI and query processing on probabilistic databases were developed independently, and their connection was established only recently [Gribkoff et al., 2014b].

This is a survey on probabilistic databases and query evaluation. The goals of this survey are the following.

1. Introduce the general independence-based data model of probabilistic databases that has been foundational to this field, and the tuple-independence formulation in particular.

2. Nevertheless, show that richer representations of probabilistic data, including soft constraints and other dependencies between the data can be reduced to the simpler tuple-independent data model. Indeed, many more knowledge representation formalisms are supported by probabilistic query evaluation algorithms, including representations from statistical relational machine learning and probabilistic programming.

3. Discuss how the semantics based on tuple independence is a convenience for building highly efficient query processing algorithms. The problem reduces to a basic reasoning task called weighted model counting. We illustrate how probabilistic query processing can benefit from the relational structure of the query, even when the data is probabilistic.

4. Discuss theoretical properties of query evaluation under this data model, identifying and delineating both tractable queries and hard queries that are unlikely to support efficient evaluation on probabilistic data.

5. Finally, provide a brief overview of practical applications and systems built on top of this technology.

The survey is organized as follows. Chapter 2 defines the probabilistic data model and presents the connection to statistical relational models. Chapter 3 discusses weighted model counting and the connection to query evaluation on probabilistic databases. Chapters 4 and 5 cover lifted inference, and query compilation respectively, showing the limitations of weighted model counting algorithms for query evaluation on probabilistic databases. Chapter 6 discusses some systems and applications of probabilistic databases.

The survey is designed to extend and complement the book on probabilistic databases by Suciu et al. [2011]. It includes material that has been developed since the publication of the book, and covers deeper the connection between probabilistic databases and weighted model counting. The discussion in Chapter 2 relating soft constraints and complex correlations to tuple-independent probabilistic databases is entirely new, as is most of the background on weighted model counting in Chapter 3. The material on lifted inference in Chapter 4 has been updated with several recent results: on non-repeating relational algebra expressions, on negation and resolution, and on symmetric databases. Chapter 5 on query compilation also includes new results on the impossiblity of computing the probability of some queries using DPLL-based algorithms, even though the queries are tractable. This survey does not cover other uncertain data models,

some of which model more complex data types, continuous variables, or even probabilistic computation. However, several of these alternatives and extensions will be reviewed in §2.7.

Notation Used Throughout the Survey

- A domain: D.
- A relational schema: $\mathbf{R}$
- A single tuple: t
- All ground tuples (Herbrand base): $\mathrm{Tup}(\mathbf{R}, D)$ or Tup
- Classical deterministic database; a set of tuples: ω or $\mathbf{T}$
- A probabilistic database: $\mathbf{D}$
- A single possible world: ω or $\mathbf{T}$.
- A set of possible worlds: $\mathbf{\Omega}$.
- Marginal probability of an individual tuple: $p(.)$
- Joint probability distributions: $\mathbf{P}(.)$
- Weight of an individual tuple: $w(.)$
- Weight of a world: $\mathbf{W}(.)$
- A specific tuple/atom: $\mathsf{Researcher}(\texttt{Bob}, \texttt{Vision}, x)$
- An attribute: `Name`, `Expertise`
- A constraint, first-order sentence or formula: Δ
- A constrained probabilistic database: $\mathbf{C}$

2

Probabilistic Data Model

Probabilistic databases model uncertain data. For example, the value of an attribute may be uncertain; or the presence/absence of an entire tuple may be known only with some probability. These are called attribute-level, and tuple-level uncertainty. For example, if we build a large database of researchers obtained by extracting author names from a large repository of journal articles, then we may obtain conflicting affiliations for a particular author; or we may not know with certainty if a particular researcher should exist in the database or not. A probabilistic database models these uncertainties using the possible worlds semantics: the state of the entire database is not known with certainty, instead we have a probability distribution on all possible instances. This chapter describes the possible world and query semantics that underlie the probabilistic data model, and explain how they can capture both attribute-level and tuple-level uncertainties. We discuss various dependence and independence assumptions, and reductions between probabilistic database models.

2.1 Possible Worlds Semantics

We fix a *database schema* $\mathbf{R} = (\mathsf{R}_1, \mathsf{R}_2, \ldots, \mathsf{R}_\ell)$ where each relation name R_j has fixed arity $r_j \geq 0$. Fix a domain D of constants. A ground tuple for relation R_j is an expression of the form $\mathsf{R}_j(\mathtt{a}_1, \ldots, \mathtt{a}_{r_j})$, where $\mathtt{a}_1, \ldots, \mathtt{a}_{r_j}$ are constants in the domain. We write $\mathrm{Tup}(\mathbf{R}, D)$ for the set of all ground tuples over the relations in $\mathbf{R}$ and domain D. A *database instance* is a subset of tuples $\mathbf{T} \subseteq \mathrm{Tup}(\mathbf{R}, D)$. We drop $\mathbf{R}$ or D from $\mathrm{Tup}(\mathbf{R}, D)$ when they are clear from context.

A (traditional) database instance $\mathbf{T}$ completely specifies the state of our knowledge about the world: for every grounded tuple we know precisely whether the tuple belongs or does not belong to the database, an assumption that is called the Closed World Assumption. In contrast, an *incomplete* database models a scenario in which the state of the database is not completely known. Instead, we talk about a set of *possible worlds* $\mathbf{\Omega} = \{\omega_1, \ldots, \omega_N\} \subseteq 2^{\mathrm{Tup}(\mathbf{R}, D)}$, where each possible world ω_i corresponds to a single database instance $\mathbf{T}_i$ (i.e., a subset of tuples). The state of the world may be one of several possible instances $\mathbf{\Omega}$. A *probabilistic* database further associates a probability with each world: we do not know which one is the actual instance, we only know the probability of each instance being the true state of the world.

Definition 2.1. A *probabilistic database*, $\mathbf{D}$, is a probability space $(\mathbf{\Omega}, \mathbf{P})$. The space of outcomes is a set of possible worlds, $\mathbf{\Omega} = \{\omega_1, \ldots, \omega_N\}$, where each ω_i is a database instance, and $\mathbf{P} : \mathbf{\Omega} \to [0, 1]$ is a probability function, i.e., $\sum_i \mathbf{P}(\omega_i) = 1$.

This is a very general and flexible definition. Each of the instances $\omega_1, \ldots, \omega_N$ is possible, and we have the freedom to set their probabilities arbitrarily, as long as they sum to one. For example, Figure 2.1 depicts four possible worlds, $\omega_1, \ldots, \omega_4$, for a relation denoting researcher affiliation, and three different probability functions, each of which sums to one. Probability function $\mathbf{P}_a$ defines a probabilistic database over these four possible worlds: $\mathbf{P}_a(\omega_1) + \cdots + \mathbf{P}_a(\omega_4) = 1$. Similarly, $\mathbf{P}_b$ or $\mathbf{P}_c$ induce different probabilistic databases over the same four possible worlds.

ω_1
Researcher

Alice	Pixar
Carol	UPenn

$\mathbf{P}_a(\omega_1) = 0.10$
$\mathbf{P}_b(\omega_1) = 0.14$
$\mathbf{P}_c(\omega_1) = 0.20$

ω_2
Researcher

Alice	Pixar
Carol	INRIA

$\mathbf{P}_a(\omega_2) = 0.10$
$\mathbf{P}_b(\omega_2) = 0.06$
$\mathbf{P}_c(\omega_2) = 0.30$

ω_3
Researcher

Alice	Brown
Carol	UPenn

$\mathbf{P}_a(\omega_3) = 0.60$
$\mathbf{P}_b(\omega_3) = 0.56$
$\mathbf{P}_c(\omega_3) = 0.40$

ω_4
Researcher

Alice	Brown
Carol	INRIA

$\mathbf{P}_a(\omega_4) = 0.20$
$\mathbf{P}_b(\omega_4) = 0.24$
$\mathbf{P}_c(\omega_4) = 0.10$

Figure 2.1: Four possible worlds $\omega_1, \ldots, \omega_4$ of a schema with one relation (Researcher) denoting research affiliations. Worlds are labeled with three different valid probability functions, $\mathbf{P}_a$, $\mathbf{P}_b$ and $\mathbf{P}_c$. Worlds not shown have probability 0.

Readers familiar with incomplete databases will note that the collection of possible worlds is precisely an *incomplete database* [Imielinski and Lipski, 1984]. In other words, a probabilistic database is an incomplete databases plus a probability distribution.

2.2 Independence Assumptions

In practice, it is impossible to enumerate and assign a probability to all possible worlds, as is done in Figure 2.1. The set of possible worlds consists of all subsets of tuples, $\omega \subseteq \mathrm{Tup}$. The number of tuples in Tup can easily be millions to billions, and the number of possible worlds is exponentially larger. Therefore, we need a specification formalism that allows us to describe the probabilistic database concisely. A common solution to this problem is to fix a probability distribution by stating simple properties of $\mathbf{P}$ that are believed to hold and that allow for a concise specification of $\mathbf{P}$. Next, we will describe two such assumptions that are commonly used. They naturally model the two types of uncertainty most often found in large datasets: tuple-level and attribute-level uncertainty.

A first type of assumption is to state the *marginal probability* that a single tuple is contained in the database. For example, we may believe that `Alice` works for `Pixar` with probability 0.2. Formally, this event is $\mathsf{Researcher}(\texttt{Alice}, \texttt{Pixar}) \in \omega$, and abusing notation, we write its marginal probability assumption as $p(\mathsf{Researcher}(\texttt{Alice}, \texttt{Pixar})) = 0.2$. Here, $p(t)$ refers to the marginal probability of a single tuple t in

the larger joint distribution $\mathbf{P}$. Similarly, we may want to state that $p(\text{Researcher}(\texttt{Carol}, \texttt{UPenn})) = 0.7$. These assumptions do not hold for $\mathbf{P}_c$ in Figure 2.1, because $p_c(\text{Researcher}(\texttt{Alice}, \texttt{Pixar})) = 0.2 + 0.3 = 0.5$. This eliminates probability function $\mathbf{P}_c$ from consideration entirely. However, the stated assumptions do hold for both $\mathbf{P}_a$ and $\mathbf{P}_b$. Hence, while intuitive and useful, marginal probability assumptions are insufficient to uniquely specify a probabilistic database.

A second type of assumption states that (subsets of) tuples are *marginally independent*. Suppose that the set of all possible tuples Tup is partitioned into k disjoint subsets $\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_k$. Intuitively, the independence assumption says that knowing a tuple from $\mathbf{T}_i$ is in the possible world does not give us any information on whether a tuple from $\mathbf{T}_j, j \neq i$, is in the possible world. Formally, it means that the probability of a possible world factorizes as

$$\mathbf{P}(\omega) = \mathbf{P}(\omega_1 \cup \omega_2 \cup \cdots \cup \omega_k) = \prod_{i=1}^{k} \mathbf{P}_{\mathbf{T}_i}(\omega_i). \tag{2.1}$$

Each ω_i is the subset of partition $\mathbf{T}_i$ found in world ω (i.e, $\omega_i = \omega \cap \mathbf{T}_i$), and $\mathbf{P}_{\mathbf{T}_i}$ denotes a joint distribution over the tuples in partition $\mathbf{T}_i$, after all other tuples have been marginalized out. We will drop subscript $\mathbf{T}_i$ when it is clear from context and write $\mathbf{P}(\omega) = \prod_{i=1}^{k} \mathbf{P}(\omega_i)$.

This type of marginal independence assumption further narrows down the set of distributions under consideration. For example, if we assume that $\text{Researcher}(\texttt{Alice}, \texttt{Pixar})$ and $\text{Researcher}(\texttt{Carol}, \texttt{UPenn})$ are independent in Figure 2.1, we can eliminate $\mathbf{P}_a$, where this assumption is false, and retain $\mathbf{P}_b$ as the intended probability function for this probabilistic database.

Next, we will discuss two more specific independence assumptions that, together with marginal probabilities of the first type, compactly and uniquely specify a probabilistic database. These assumptions are most popular in the literature, since they correspond to common types of uncertainty in data: block-independent disjoint and tuple-independent probabilistic databases.

2.2.1 Block-Independent Disjoint Databases

The most common type of uncertainty is when the value of an attribute is not known precisely. This is also called attribute-level uncertainty. The SQL standard uses NULL to record the fact that a value is missing. When the value is unknown, but some information is known about it, SQL offers no mechanism for representing the partial information about that value. Quite often, we can describe a probability distribution on the possible missing values; a probabilistic database allows us to store that probability distribution.

Consider for example the database instance in Figure 2.2. The table stores three researchers whose areas of expertise are known, but whose affiliations are not known with precision. Instead, for each researcher there is a probability distribution on the possible affiliations where she/he might be. For `Alice` there are two possible affiliations, for `Bob` there are three, and for `Carol` there are two. There are twelve possible combinations, and each combination gives rise to a possible world. Therefore, our probabilistic database is a probability distribution over twelve possible worlds.

Researcher

Name	Expertise	Affiliation	
Alice	Graphics	Pixar	0.3
		Brown	0.7
Bob	Vision	UPenn	0.3
		PSU	0.3
		Brown	0.4
Carol	Databases	UPenn	0.5
		INRIA	0.5

Figure 2.2: Attribute-Level Uncertainty

In practice, a reasonable assumption is that the affiliations of these three people are independent probabilistic events. In that case, the probability of a possible world is the product of three probabilities, one for each of the selected values. Figure 2.3 shows this distribution, where we drop the `Expertise` attribute to reduce clutter.

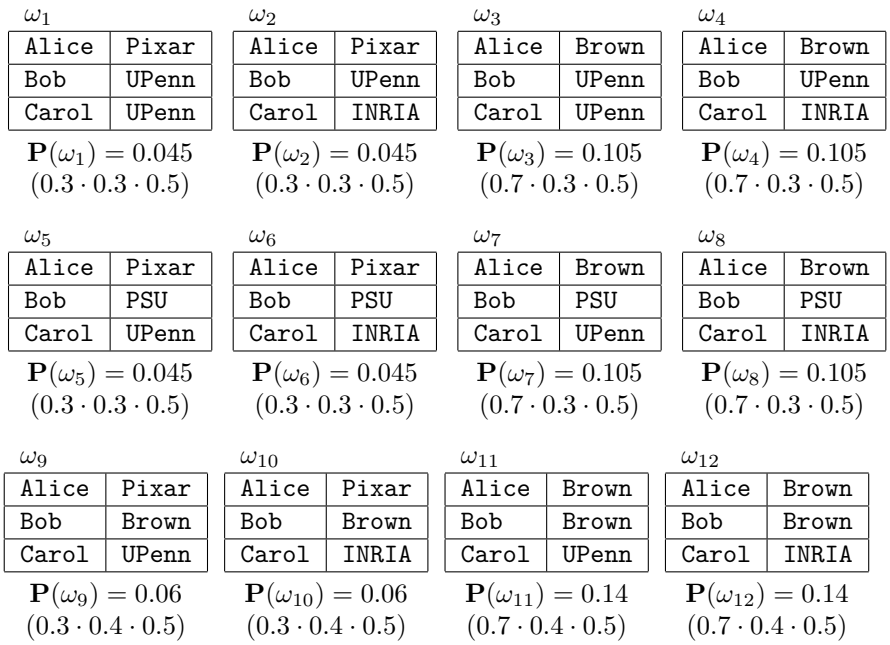


Figure 2.3: Possible worlds for the Probabilistic Database in Figure 2.2.

Attribute-level uncertainty occurs often in practice, and was used in the earliest references to probabilistic databases. For example, Barbará et al. [1992] allow attribute values to be associated with probabilities, and also allow the attribute value be completely unspecified (NULL), with a given probability. Orion [Singh et al., 2008] goes a step further and allows the value of an attribute to be a continuous distribution, by storing the parameters of a probability density function from a predefined library (e.g., Binomial, Poisson, etc.).

Trio [Benjelloun et al., 2006a] defines an *X-tuple* to be a set of tuples together with a probability distribution over the elements of the set. For example, the set of tuples $\mathsf{Researcher}(\mathtt{Bob}, \mathtt{Vision}, \mathtt{UPenn})$, $\mathsf{Researcher}(\mathtt{Bob}, \mathtt{Vision}, \mathtt{PSU})$, and $\mathsf{Researcher}(\mathtt{Bob}, \mathtt{Vision}, \mathtt{Brown})$ with probabilities $0.3, 0.3, 0.4$ respectively forms an X-tuple. In the marginal independence assumption of Equation 2.1, the distribution $\mathbf{P}(\omega_i)$ becomes an X-tuple when we further assume that each possible world contains at most one tuple from each $\mathbf{T}_i$, or exactly one when the sum

Researcher

Name	Expertise	Affiliation	p
Alice	Graphics	Pixar	0.3
Alice	Graphics	Brown	0.7
Bob	Vision	UPenn	0.3
Bob	Vision	PSU	0.3
Bob	Vision	Brown	0.4
Carol	Databases	UPenn	0.5
Carol	Databases	INRIA	0.5

Figure 2.4: Block-Independent Disjoint (BID) table for the relation in Figure 2.2.

of all probabilities in each $\mathbf{T}_i$ is $= 1$. Thus, the semantics is that one alternative is included independently for all X-tuples in the relation. Together, a set of X-tuples for the same relation form an X-relation. The relation in Figure 2.2 can be seen as an X-relation consisting of three X-tuples. X-relations generalize attribute-level uncertainty, because they can define distributions on two or more attributes. In an X-relation, the grouping of tuples into X-tuples needs to be specified using *lineage*.

Ré and Suciu [2007] introduce *Block-Independent Disjoint Tables* or BID tables, which are X-relations where the grouping of tuples is defined by a set of attributes, called a key, and the probability is stored explicitly as a separate, distinguished attribute p. For example, the relation in Figure 2.2 can be represented by the BID table in Figure 2.4. The key consists of the attributes `Name` and `Expertise`, which are underlined in the table. They partition the tuples into blocks $\mathbf{T}_i$: within each block the tuples are *disjoint*, while across blocks the tuples are *independent*, hence the name BID. Within each block the sum of all probabilities must be ≤ 1. In any possible world, the pair of attributes $(\texttt{Name}, \texttt{Expertise})$ forms a key in the traditional database sense.

2.2.2 Tuple-Independent Databases

A second kind of database uncertainty is tuple-level uncertainty, where the existence of an entire tuple is unknown. For example, an information extraction system crawls a large text corpus (e.g. a Web crawl) and extracts entities or relationships between entities [Etzioni

et al., 2008, Mitchell et al., 2015]. The extractions are performed using statistical machine learning, and therefore inherently uncertain. Every extracted tuple has a degree of confidence associated with it.

A tuple-independent relation is represented by a standard database instance where each relation has a distinguished attribute storing the marginal probability of that tuple. Formally, a tuple-independent probabilistic database is a pair $\mathbf{D} = (\mathbf{T}, p)$, where $\mathbf{T}$ is a standard database (a set of tuples) and $p : \mathbf{T} \to [0, 1]$ associates a probability to each tuple in $\mathbf{T}$. Any subset of tuples forms a possible world, obtained by including randomly and independently each tuple $t \in \mathbf{T}$, with the probability specified in the database, $p(t)$. Worlds that contain tuples not found in $\mathbf{T}$ have probability zero. We denote by $\mathbf{P_D}$ the probability induced by the tuple-independent database $\mathbf{D}$:

$$\mathbf{P_D}(\omega) = \begin{cases} \prod_{t \in \omega} p(t) \prod_{t \in \mathbf{T} - \omega} (1 - p(t)) & \text{if } \omega \subseteq \mathbf{T} \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

This data model captures the marginal independence assumption of Equation 2.1 where each partition $\mathbf{T}_i$ consists of a single tuple $t \in \mathbf{T}$. We drop the subscript and simply write $\mathbf{P}(\omega)$ when the probabilistic database is clear from the context. In practice, we represent $\mathbf{D}$ by simply extending the schema of $\mathbf{T}$ to include the probability p as an extra attribute.

For example, Figure 2.5a shows a hypothetical table extracted from the Web, consisting of (`CEO`,`Company`) pairs. The extractor is not fully confident in its extractions, so that the tuple Manager(`David`, `PestBye`) has a confidence level of only 60%, while the tuple Manager(`Elga`, `KwikEMart`) has a confidence level of 90%. Any subset of the uncertain tuples is a possible world, hence there are eight possible worlds (not shown). Here, too, the simplest way to uniquely define the probability function is to assume independence. In that case the probability of any possible world is the product of the probabilities of the tuples in the world, times the product of one minus the probabilities of the tuples not in the world. For example, the probability of the world {Manager(`David`, `PestBye`), Manager(`Fred`, `Vulgari`)} is $0.6 \cdot (1 - 0.9) \cdot 0.8 = 0.048$.

Manager

CEO	Company	p
David	PestBye	0.6
Elga	KwikEMart	0.9
Fred	Vulgari	0.8

(a) Probabilities

Manager

CEO	Company	w
David	PestBye	1.5
Elga	KwikEMart	9.0
Fred	Vulgari	4.0

(b) Weights

Figure 2.5: A tuple-independent Manager relation. For each tuple we can specify the probability p as shown in (a), or the weight w as shown in (b).

2.2.3 Weights vs. Probabilities

An alternative, yet equivalent way to specify a tuple-independent relation is to assign a weight $w(t)$ to each tuple t. We refer to such databases as *tuple-independent weighted databases*. Then, each possible world has a weight $\mathbf{W_D}$ defined as the product of the weights of all tuples in that world:

$$\mathbf{W_D}(\omega) = \begin{cases} \prod_{t \in \omega} w(t) & \text{if } \omega \subseteq \mathbf{T} \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

Figure 2.5b illustrates a probabilistic database represented with weights w instead of probabilities p.

To obtain the probability of a possible world from its weight, we divide by a normalization factor Z:

$$\mathbf{P_D}(\omega) = \frac{\mathbf{W_D}(\omega)}{Z} \qquad \text{where } Z = \sum_{\omega \subseteq \mathbf{T}} \mathbf{W_D}(\omega). \tag{2.4}$$

We leave it to the reader as an exercise to check that Z is a simple product, $Z = \prod_{t \in \mathbf{T}}(1 + w(t))$, by virtue of tuple-independence.

A probabilistic database represented in terms of tuple probabilities is equivalent to a probabilistic database represented in terms of weights, in the following sense. Define the weight of each tuple t as its *odds*, that is, $w(t) = p(t)/(1 - p(t))$. Then, it is easy to check that Equations 2.2 and 2.4 are equivalent. Conversely, by setting probabilities $p(t) = w(t)/(1 + w(t))$ we can turn weights into probabilities.

The probabilistic databases in Figures 2.5a and 2.5b are indeed equivalent.

Probabilities are a more natural and intuitive representation than weights for tuple-independent probabilistic relations. We will explain the rationale for considering weights in §2.5, when we introduce soft constraints and Markov Logic Networks.

2.3 Query Semantics

Building on the possible world semantics, this section studies the semantics of a query over probabilistic databases: given a query Q in some query language, such as SQL, datalog, or relational calculus, what should Q return on a probabilistic database? Query semantics on a traditional database is defined by some sort of induction on the structure of the query expression; for example, the value of a relational algebra expression is defined bottom up. Our semantics over probabilistic databases is different, in the sense that it ignores the query expression, and instead assumes only that the query already has a well-defined semantics over deterministic databases. Our task will be to extend this semantics to probabilistic databases. In other words, assuming we know exactly how to compute Q on a traditional database $\mathbf{T}$, we want to define the meaning of Q on a probabilistic database $\mathbf{D}$.

It is convenient to assume that the semantics of a query Q over traditional databases is a mapping from database instances (2^{Tup}) into d-dimensional vectors ($\mathbb{R}^d$). Then, its semantics over probabilistic databases is simply its expectation, which is a vector as well.

Definition 2.2. A query Q over a traditional database is a function $Q : 2^{\text{Tup}} \to \mathbb{R}^d$. The semantics of Q over a probabilistic database $\mathbf{D} = (\mathbf{\Omega}, \mathbf{P})$ is the expectation $\mathbf{E}_{\mathbf{D}}[Q] = \sum_{\omega \in \mathbf{\Omega}} \mathbf{P}_{\mathbf{D}}(\omega) \cdot Q(\omega)$.

Note that this definition is query-language agnostic: it supports relational calculus, algebra, or datalog. It only requires that the query language is defined on traditional databases, and that query answers are vectors. However, it is important to realize that this is only a definition, and says nothing about how to actually compute the query

efficiently over a probabilistic database. We will address the *query evaluation* problem in Chapters 3, 4, and 5, and see that there are important algorithmic differences, depending on which query language is used.

We end this discussion with a brief review of standard query expressions. A *Conjunctive Query* (CQ) is an existentially quantified conjunction of positive relational atoms: the free (unquantified) variables $\mathbf{x}$ are sometimes called head variables. A *Union of Conjunctive Queries* (UCQ) is a disjunction of conjunctive queries with the same head variables. Formally,

$$\begin{aligned} CQ(\mathbf{x}) = &\ \exists y_1 \cdots \exists y_k,\ \mathsf{R}_1(\tau_1) \wedge \cdots \wedge \mathsf{R}_\ell(\tau_\ell) \\ UCQ(\mathbf{x}) = &\ CQ_1(\mathbf{x}) \vee \cdots \vee CQ_m(\mathbf{x}), \end{aligned}$$

where each τ_j is a sequence of terms (logical variables or constants).

Example Query Semantics

Next, we illustrate with several examples how this broad definition of query semantics applies to different types of queries.

Boolean Queries A Boolean query is a function $Q : 2^{\mathrm{Tup}} \to \{0, 1\}$. Hence, $d = 1$, and $\mathbf{E}[Q]$ is the probability that Q is true on a random database instance. We will use the notations $\mathbf{E}[Q]$ and $\mathbf{P}(Q)$ interchangeably. Consider for example the following Boolean query, written as a datalog rule.

$$Q1 \text{ :- } \mathsf{Researcher}(x, y, \texttt{Brown}).$$

The query asks: *Is there any researcher affiliated with* `Brown`*?* Given a traditional (deterministic) database $\mathbf{T}$, the answer to the query is true or false, that is, 1 or 0. Given a probabilistic database, its answer is the sum of probabilities of all possible worlds where the query is true. For a simple example, the reader may verify that the query's answer on the BID database in Figure 2.2 is $\mathbf{P}(Q1) = 1 - (1 - 0.7)(1 - 0.4) = 0.82$. Recall that we assume `Alice`'s affiliation to be independent of `Bob`'s.

Set-Valued Queries Typical relational queries return a set of tuples. We will call these set-valued queries, to distinguish from Boolean or

aggregate queries. Each query has an arity r, which is the number of head variables involved in the answer. Assuming a finite domain D, there are at most $d = |D|^r$ possible tuples in the answer. A set-valued query can be seen as a function $Q : 2^{\text{Tup}} \to \{0,1\}^d$. For example, the following query returns all expertises at `Brown`.

$$Q2(y) \text{ :- } \mathsf{Researcher}(x, y, \texttt{Brown}).$$

The query has arity $r = 1$. Since there are only three distinct expertises in the domain of the database in Figure 2.2, we can assume $d = 3$, and the expected value vector $\mathbf{E}[Q2]$ is

`Graphics`	0.7
`Vision`	0.4
`Databases`	0.0

Aggregate Queries SQL queries with group-by and aggregates can also be captured by Definition 2.2. For example, consider the aggregate query: *Count the number of researchers for each affiliation*. In datalog, this query is written as

$$Q3(z, \mathit{count}(*)) \text{ :- } \mathsf{Researcher}(x, y, z).$$

The answer vector has dimension $d = 5$, because there are five possible affiliations in Figure 2.2. The expected value vector $\mathbf{E}[Q3]$ is

`Pixar`	0.3
`Brown`	1.1
`UPenn`	0.8
`PSU`	0.3
`INRIA`	0.5

The second expected count, of 1.1 researchers at `Brown`, is easily verified from Figure 2.3. The probability of all possible worlds where `Brown` has exactly one researcher is $0.105 \cdot 4 + 0.06 \cdot 2 = 0.54$. The probability it has exactly two researchers is $2 \cdot 0.14 = 0.28$. This gives an expected number of $0.54 \cdot 1 + 0.28 \cdot 2 = 1.1$ researchers.

2.4 Beyond Independence: Hard Constraints

In standard, deterministic databases, a hard constraint is an assertion that must hold on any database instance. In probabilistic databases, hard constraints are a convenient instrument to encode dependencies between probabilistic tuples, by restricting possible worlds.

2.4.1 Types of Constraints

Typical constraints expressed in SQL are key constraints, foreign key constraints, or conditions on attribute values. A general class of constraints studied in databases consists of sentences Δ of the form

$$\forall \mathbf{x}\, \varphi(\mathbf{x}) \Rightarrow \exists \mathbf{y}\, \psi(\mathbf{x}, \mathbf{y}),$$

where each of φ and ψ is a conjunction of positive atoms, that is, a conjunction of relational atoms $\mathsf{R}(x_1, x_2, \ldots)$ or equality predicates $x = y$. Such a constraint is called a *Generalized Dependency* (GD), or a Generalized Constraint. Two examples used in the literature are Local-As-View and Global-As-View. The GD is a *Local-As-View* (LAV) if φ consists of a single relational atom and every variable in $\mathbf{x}$ occurs in some relational atom in ψ. When the constraint has no existential quantifiers and ψ consists of a single relational atom, it is called a *Global-As-View* (GAV) constraint. Other two standard examples used in the literature are equality-generating constraints and tuple-generating constraints. In an equality-generating constraint, ψ consists only of equality predicates; in a tuple generating constraint, ψ consists only of relational atoms. We illustrate one of each, over relations $\mathsf{Researcher}(\texttt{Name}, \texttt{Expertise}, \texttt{Affiliation})$ and $\mathsf{University}(\texttt{UName}, \texttt{City})$:

$$\begin{aligned} \forall x, y_1, z_1, y_2, z_2,\ \mathsf{Researcher}(x, y_1, z_1) \wedge \mathsf{Researcher}(x, y_2, z_2) \\ \Rightarrow y_1 = y_2 \wedge z_1 = z_2 \\ \forall x, y, z,\ \mathsf{Researcher}(x, y, z) \Rightarrow \exists c\, \mathsf{University}(z, c). \end{aligned}$$

The first constraint is a key constraint, stating that `Name` is a key in Researcher. The second constraint is an inclusion constraint, specifying that every affiliation must be a university.

2.4.2 Constrained Probabilistic Databases

To enforce constraints over all possible worlds, a constrained probabilistic database sets the probability of worlds that violate Δ to zero.

Definition 2.3. A *constrained probabilistic database* $\mathbf{C} = (\mathbf{D}, \Delta)$ consists of a probabilistic database $\mathbf{D}$ and a constraint Δ. It represents the probability distribution:

$$\mathbf{P}_{\mathbf{C}}(\omega) = \mathbf{P}_{\mathbf{D}}(\omega) \cdot \Delta(\omega) / \mathbf{P}_{\mathbf{D}}(\Delta). \tag{2.5}$$

Here, $\Delta(\omega)$ evaluates Δ as a Boolean query on database ω, returning either true (1) or false (0).

In other words, the probability of each world is its conditional probability $\mathbf{P}_{\mathbf{D}}(\omega|\Delta)$ in the unconstrained probabilistic database. Query semantics are defined identically to the unconstrained case, as the expectation of Q, now over the constrained distribution:

$$\mathbf{E}_{\mathbf{C}}[Q] = \sum_{\omega \in \mathbf{\Omega}} \mathbf{P}_{\mathbf{C}}(\omega) \cdot Q(\omega) = \mathbf{E}_{\mathbf{D}}[Q \cdot \Delta] / \mathbf{P}_{\mathbf{D}}(\Delta).$$

For Boolean queries specifically, we have that $\mathbf{P}_{\mathbf{C}}(Q) = \mathbf{P}_{\mathbf{D}}(Q \wedge \Delta) / \mathbf{P}_{\mathbf{D}}(\Delta) = \mathbf{P}_{\mathbf{D}}(Q|\Delta)$.

A BID table satisfies its key constraint Δ by definition. However, it is not equivalent to a tuple-independent probabilistic table constrained by that same Δ. To see the difference, consider a BID table $\mathsf{Researcher}_1(\underline{\texttt{Name}}, \texttt{Affiliation})$ and a tuple-independent table for the same relation $\mathsf{Researcher}_2(\texttt{Name}, \texttt{Affiliation})$, with identical probabilities, but without `Name` being a key.

$\mathsf{Researcher}_1$

Name	Affiliation	p
Alice	Pixar	0.5
Alice	Brown	0.5

$\mathsf{Researcher}_2$

Name	Affiliation	p
Alice	Pixar	0.5
Alice	Brown	0.5

The BID table $\mathsf{Researcher}_1$ represents two possible worlds with non-zero probability, each with probability 0.5. The independent table $\mathsf{Researcher}_2$ represents four possible worlds (one for each subset), each with probability 0.25. Conditioned on the key constraint $\Delta = (\forall x, y_1, y_2, \mathsf{Researcher}_2(x, y_1) \wedge \mathsf{Researcher}_2(x, y_2) \Rightarrow y_1 = y_2)$ on `Name`, the tuple-independent probabilistic database has three possible worlds (including the empty world) with probability $1/3$ each.

2.4.3 Queries on Constraints

We identify two types of queries that become natural when a probabilistic database is subject to constraints.

Consistency Degree Each constraint Δ is also a Boolean query, being true or false in every traditional database, and having a probability in every probabilistic database. Constraints play a simple role in probabilistic databases as a metric of quality. Consistent possible worlds satisfy the constraint, and inconsistent worlds violate it. Our semantics for Boolean queries supports asking for the probability $\mathbf{P}(\Delta)$ on a probabilistic database. It represents the degree to which a random world is consistent. If the probabilistic database is tuple-independent, or BID, then this probability is astronomically small, since a constraint is a universally quantified sentence and its probability in a random world is very small. For example, consider a key constraint Δ over a tuple-independent relation. Suppose that the key attribute has n values, $\mathtt{K}_1, \mathtt{K}_2, \ldots, \mathtt{K}_n$, and for each value $\mathtt{K}_i$ there are two conflicting tuples with that key, both with probability p. Since these tuples are independent, the probability that a random world satisfies the key constraint is $\mathbf{P}(\Delta) = (1 - p^2)^n$. For typical values $p = 0.5$ and $n = 10^6$, we have that $\mathbf{P}(\Delta) = 10^{-125000}$. Nevertheless, consistency queries can be useful to estimate the relative degree of consistency between databases and constraints.

Most-Probable Database Given a belief in the correctness of each tuple, in the form of a probabilistic database $\mathbf{D}$, and given key or dependency constraints Δ, a natural task in database repair is to recover the true database where all constraints are satisfied. The *most-probable database* (MPD) query solves this task [Gribkoff et al., 2014c]. It finds the most-probable world ω, according to $\mathbf{P}_\mathbf{D}$, where the constraints are satisfied, or equivalently, in $\mathbf{P}_\mathbf{C}$ for $\mathbf{C} = (\mathbf{D}, \Delta)$. Formally,

$$\mathbf{MPD}_\mathbf{D}(\Delta) = \arg\max_{\omega \in \mathbf{\Omega}} \mathbf{P}_\mathbf{D}(\omega) \cdot \Delta(\omega) = \arg\max_{\omega \in \mathbf{\Omega}} \mathbf{P}_\mathbf{C}(\omega). \tag{2.6}$$

The MPD query can be used to answer *most-probable explanation* (MPE) or *maximum a posteriori probability* (MAP) queries in more expressive relational data models.

Next, we discuss an application of constrained probabilistic databases: to encode soft constraints and Markov Logic Networks.

2.5 Beyond Independence: Soft Constraints

Most probabilistic models used in probabilistic databases are limited to independent tuples, or disjoint and independent tuples. This simplifies the study of probabilistic inference, since query evaluation is simply weighted model counting, as we explain in the next chapter. However, most applications require a richer probability model where the random variables corresponding to the tuples in the database are correlated in complex ways. It turns out that such correlations can be captured in probabilistic databases through soft constraints, using the semantics of Markov Logic Networks (MLN). We describe MLNs here, and show how they can be used as soft constraints over probabilistic databases. Then, in the next section we show how soft constraints can be rephrased as conditional probabilities over tuple-independent databases, allowing us to both model complex correlations, and still have a simple definition of the query's semantics in terms of weighted model counting. The takeaway of this section is that query evaluation on complex probabilistic dependencies is equivalent to computing the conditional probability over a tuple-independent database, whose conditional is given by a constraint (to be formally defined in §2.6.2).

2.5.1 Markov Logic Networks

Markov Logic Networks (MLNs), introduced by Richardson and Domingos [2006], are the simplest of a class of statistical relational models [Getoor and Taskar, 2007, De Raedt et al., 2016], which aim to represent a probability distribution over relational databases. We give here a brief overview and refer the reader to Domingos and Lowd [2009] for more details on MLNs. Our definitions are a slight departure from those in Richardson and Domingos [2006] or Domingos and Lowd [2009], as we will explain in §2.5.3.

An MLN is a collection of soft constraints, consisting of a first-order formula, and a weight. Before giving the formal definition, we

illustrate MLNs with an example. Consider the following two soft constraints.

$$3.5 \quad \mathsf{Smoker}(x) \wedge \mathsf{Friend}(x, y) \Rightarrow \mathsf{Smoker}(y)$$
$$1.3 \quad \mathsf{Smoker}(x) \Rightarrow \mathsf{Cancer}(x)$$

The first constraint says that, typically, friends of smokers are also smokers, while the second constraint says that smokers are at risk of developing cancer. The weight of each constraint indicates how strongly that constraint should hold in a possible world: the higher the weight, the more confident we are in the constraint.

Formally, an MLN, M, is a set of pairs

$$M = \{(w_1, \Delta_1(\mathbf{x}_1)), (w_2, \Delta_2(\mathbf{x}_2)), \ldots\}$$

where each $\Delta_i(\mathbf{x}_i)$ is a first-order formula whose free (unquantified) variables are $\mathbf{x}_i$. We refer to each $\Delta_i(\mathbf{x}_i)$ as a constraint and to each pair $(w_i, \Delta_i(\mathbf{x}_i))$ as a soft constraint.

Semantics

A first-order *sentence* is a first-order formula without free variables, that is, one where each logical variable is associated with a universal or existential quantifier. For a given finite domain D, a first-order formula $\Delta(\mathbf{x})$ can be seen as representing a set of first-order sentences $\{\delta_1, \ldots, \delta_d\}$, obtained by substituting the free variables $\mathbf{x}$ with constants from the domain D. We will refer to these sentences as the groundings of $\Delta(\mathbf{x})$. For example, a grounding of the first constraint above is $\delta = \mathsf{Smoker}(\mathtt{Alice}) \wedge \mathsf{Friend}(\mathtt{Alice}, \mathtt{Bob}) \Rightarrow \mathsf{Smoker}(\mathtt{Bob})$. We use the term grounding with some abuse, since, in general, δ may be a sentence with quantified variables. For example, if $\Delta(x) = (\mathsf{R}(x) \Rightarrow \exists y\ \mathsf{S}(x, y))$, then one of its groundings is the sentence $\delta = (\mathsf{R}(5) \Rightarrow \exists y \mathsf{S}(5, y))$. The *grounding of an MLN* M over domain D is defined as follows.

$$\mathrm{ground}(M) = \{(w_i, \delta) \mid (w_i, \Delta_i(\mathbf{x})) \in M \text{ and } \delta \text{ is a grounding of } \Delta_i(\mathbf{x})\}$$

Next, consider a single possible world $\omega \subseteq \mathrm{Tup}$. Recall that the notation $\omega \models \delta$ means that the sentence δ is true in ω.

The MLN semantics define the weight of the world ω as

$$\mathbf{W}_M(\omega) = \prod_{(w,\delta)\in \text{ground}(M):\ \omega \models \delta} w. \tag{2.7}$$

In other words, the weight of a world ω is the product of weights of all grounded constraints that are true in ω.

The MLN defines the following probability distribution over the set of possible worlds:

$$\mathbf{P}_M(\omega) = \mathbf{W}_M(\omega)/Z \qquad \text{where} \qquad Z = \sum_{\omega \subseteq \text{Tup}} \mathbf{W}_M(\omega) \tag{2.8}$$

To get some intuition behind soft constraints and their associated weights, we state the following simple facts, and invite the reader to verify them:

- A soft constraint with weight $w = 1$ means *don't care*. More precisely, if $(1, \Delta(\mathbf{x}))$ is a soft constraint in the MLN, then the probability distribution remains unchanged if we remove this soft constraint from the MLN.
- A soft constraint $(0, \Delta(\mathbf{x}))$ enforces the hard constraint $\neg\exists\mathbf{x}\Delta(\mathbf{x})$.
- Increasing the weight makes a constraint more likely. Formally, fix one grounded constraint (w, δ). Suppose we increase the weight of δ from w to $w' > w$, while keeping all other weights unchanged. Then, for any world ω that satisfies δ, its probability will increase, and for any ω that does not satisfy δ, its probability will decrease.
- Constraints $(w, \Delta(\mathbf{x}))$ and $(1/w, \neg\Delta(\mathbf{x}))$ are equivalent. Formally, if we replace one soft constraint $(w, \Delta(\mathbf{x}))$ in the MLN by $(1/w, \neg\Delta(\mathbf{x}))$, the new MLN defines a different weight function $\mathbf{W}_M$, yet exactly the same probability distribution $\mathbf{P}_M$.
- However, comparing weights of different soft constraints does not tell us anything about their probabilities. More precisely, if (w, δ) and (w', δ') are two grounded constraints and $w < w'$, this does not imply, in general, that $\mathbf{P}(\delta) \leq \mathbf{P}(\delta')$. (We invite

the reader to find an example where $\mathbf{P}(\delta) > \mathbf{P}(\delta')$.) MLNs give great flexibility in expressing dependencies, yet also a risk of being uninterpretable, because weighted constraints can partially undo each other.

2.5.2 Weighted Databases with Soft Constraints

MLNs can be used as, and are equivalent to soft constraints in probabilistic databases. Given a probabilistic database, the addition of an MLN affects the probabilities of the possible worlds, by favoring those that satisfy the soft constraints. The formal semantics of a probabilistic database with soft constraints can be obtained by viewing the database itself as a set of soft constraints, and then applying the MLN semantics of Equation 2.8.

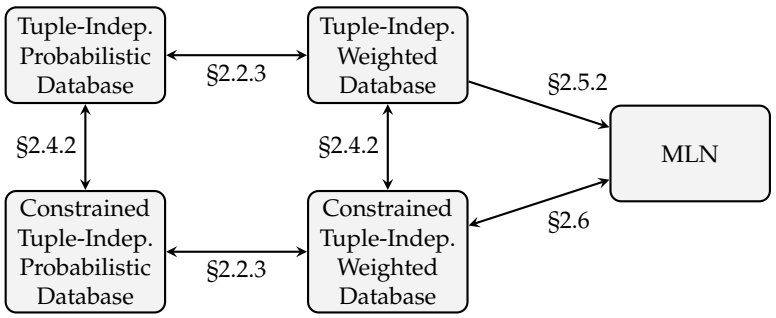


Figure 2.6: Reductions between MLNs and various probabilistic database models.

This reduction is illustrated in the upper half of Figure 2.6 (the lower half will be explained in §2.6). Take a tuple-independent probabilistic database $\mathbf{D}_p = (\mathbf{T}, p)$. As discussed in §2.2.3, this database is easily transformed into a tuple-independent weighted database $\mathbf{D}_w = (\mathbf{T}, w)$, by setting the weight of each tuple to its odds (see Figure 2.5 for an example). Next, $\mathbf{D}_w$ is transformed into the MLN

$$\{(w(t), t) \mid t \in \mathbf{T}\} \cup \{(0, t) \mid t \in \mathrm{Tup} - \mathbf{T}\}. \tag{2.9}$$

In other words, we create one soft constraint for every tuple in the database, and one soft constraint with weight 0 for every tuple not in the database. The probability distribution on possible worlds defined

by the tuple-independent database (Equation 2.4) is the same as the probability distribution defined by the associated MLN (Equation 2.8).

Definition 2.4. A weighted database with soft constraints is a pair $(\mathbf{D}, M)$ where $\mathbf{D}$ is a weighted tuple-independent database and M is a Markov Logic Network. It represents the probability distribution defined by the union of two MLNs: (1) the soft constraints encoded in $\mathbf{D}$ (Equation 2.9), and (2) the soft constraints in M.

Manager

CEO	Company	w
David	PestBye	u_{11}
David	KwikEMart	u_{12}
Elga	KwikEMart	u_{22}

Smoker

Person	w
David	r_1
Elga	r_2
Fred	r_3

$$M = \{(v, \mathsf{Manager}(x, y) \Rightarrow \mathsf{Smoker}(x))\}$$

Figure 2.7: A weighted database with one soft constraint.

For example, consider the weighted tuple-independent probabilistic database in Figure 2.7, over domain $D = \{\texttt{David}, \texttt{Elga}, \texttt{Fred}, \texttt{PestBye}, \texttt{KwikEMart}\}$. Ignore the MLN in the figure for the moment. The tuple-independent database has 2^{30} possible worlds (since there are 30 grounded tuples: 25 for Manager and 5 for Smoker), but only 2^6 possible worlds have a non-zero weight. Recall that the weight of each world is the product of the weights of its tuples. For example, the empty world has weight 1, the world consisting of all six tuples has weight $u_{11}u_{12}u_{22}r_1r_2r_3$, and the world $\omega_1 = \{\mathsf{Manager}(\texttt{David}, \texttt{KwikEMart}), \mathsf{Manager}(\texttt{Elga}, \texttt{KwikEMart}), \mathsf{Smoker}(\texttt{Fred})\}$ has weight $\mathbf{W}(\omega_1) = u_{12}u_{22}r_3$. The normalization factor of the database is

$$Z = \sum_{\omega} \mathbf{W}(\omega) = (1 + u_{11})(1 + u_{12})(1 + u_{22})(1 + r_1)(1 + r_2)(1 + r_3).$$

Let us now add the soft constraint $(v, \mathsf{Manager}(x, y) \Rightarrow \mathsf{Smoker}(x))$ to our distribution. It is a soft inclusion constraint saying *typically, values for* CEO *that occur in* Manager *also occur in* Smoker. To see how the constraint changes the weights of the possible worlds, we first need

to ground the MLN over the domain D. We obtain 25 grounded constraints in $\mathrm{ground}(M)$:

$$\{(v, \mathsf{Manager}(\mathtt{David}, \mathtt{David}) \Rightarrow \mathsf{Smoker}(\mathtt{David})),$$
$$(v, \mathsf{Manager}(\mathtt{David}, \mathtt{Elga}) \Rightarrow \mathsf{Smoker}(\mathtt{David})),$$
$$\ldots,$$
$$(v, \mathsf{Manager}(\mathtt{PestBye}, \mathtt{KwikEMart}) \Rightarrow \mathsf{Smoker}(\mathtt{PestBye})),$$
$$(v, \mathsf{Manager}(\mathtt{KwikEMart}, \mathtt{KwikEMart}) \Rightarrow \mathsf{Smoker}(\mathtt{KwikEMart}))\}.$$

The weight of each world is now computed by first multiplying the weights of all its tuples (as before), and next multiplying the weights of all grounded constraints that are true in that world. For example, the possible world ω_1 defined above satisfies 23 grounded constraints. These are all groundings, except for $\mathsf{Manager}(\mathtt{David}, \mathtt{KwikEMart}) \Rightarrow \mathsf{Smoker}(\mathtt{David})$ and $\mathsf{Manager}(\mathtt{Elga}, \mathtt{KwikEMart}) \Rightarrow \mathsf{Smoker}(\mathtt{Elga})$, which are both violated in world ω_1. Therefore the weight of ω_1 is $\mathbf{W}(\omega_1) = u_{12}u_{22}r_3v^{23}$. We invite the reader to check that the normalization factor $Z = \sum_\omega \mathbf{W}(\omega)$ no longer has a simple closed form (for non-trivial constraints), and this makes it much harder to compute the probability of a world.

Finally, note that in this probability distribution, the tuples are no longer independent. For example, the presence of the tuple $\mathsf{Manager}(\mathtt{David}, \mathtt{PestBye})$ in a possible world increases the probability of $\mathsf{Smoker}(\mathtt{David})$, which in turn increases the probability of $\mathsf{Manager}(\mathtt{David}, \mathtt{KwikEMart})$.

2.5.3 Discussion

The soft constraints in MLNs create complex correlations between the tuples in a probabilistic database. In MLNs we can also define hard constraints, by giving them a weight of $w = 0$ or $w = \infty$. In the first case we simply assert that the constraint is false, since all worlds satisfying that constraint have weight 0. In the second case, the weight of a world that satisfies the constraint becomes ∞, and then its probability is no longer well defined, since both numerator and denominator of Equation 2.8 are ∞. There are two workarounds that lead to the same

result: the first is to set w to be a finite quantity, then let $w \to \infty$ in Equation 2.8; the second is to restrict the set of possible worlds to only those that satisfy all hard constraints, and remove the weights ∞ from the product in Equation 2.7. We invite the reader to check that the two definitions are equivalent.

Our semantics of MLNs differ slightly from the original definition in Richardson and Domingos [2006]. In that definition, an MLN is a set of pairs of weights and sentences, (w, Δ), where Δ is a sentence (closed formula). The grounding of the formula is obtained by essentially treating all universally quantified variables as free variables. For example, the constraint in Figure 2.7 is written as $(w, \forall x \forall y, \mathsf{Manager}(x, y) \Rightarrow \mathsf{Smoker}(x))$, then the constraint is grounded in n^2 possible ways over a domain of size n, by essentially treating x and y as free variables. There are two problems with the original definition. The first is that existential quantifiers are not handled. Second, sentences that are semantically equivalent may lead to different interpretations. For example, $\Delta_1 = \forall x \forall y \mathsf{Researcher}(x) \wedge \mathsf{Smoker}(y)$ and $\Delta_2 = \forall x \mathsf{Researcher}(x) \wedge \mathsf{Smoker}(x)$ are logically equivalent sentences, yet the first has n^2 groundings while the second has n groundings, leading to rather different probability distributions. We corrected this inconsistency by requiring the formula to be given with explicit free variables. The soft constraints now become $\Delta_1(x, y) = \mathsf{Researcher}(x) \wedge \mathsf{Smoker}(y)$ and $\Delta_2(x) = \mathsf{Researcher}(x) \wedge \mathsf{Smoker}(x)$, which are no longer equivalent.

The reader may wonder why MLNs use weights instead of probabilities. We have argued earlier that weights and probabilities are interchangeable for the purpose of defining tuple-independent databases; however, they are no longer interchangeable for soft constraints. Given a choice, we would prefer to specify probabilities: weights have no intuitive semantics, except for the simple fact that, if we increase the weight of one soft constraint while keeping all other weights fixed, then we also increase its marginal probability. One may wonder what happens if we defined soft constraints using marginal probabilities instead of weights. For example, consider a collection of soft constraints defined as pairs (p, Δ), where p is interpreted as the

marginal probability of Δ. Then it is difficult to define a probability distribution over all possible worlds that is consistent with all these marginal probabilities (i.e., the probability of Δ is indeed p, as specified). Such a distribution may not be unique, or may not exist at all if the probabilities are inconsistent. For example, the two soft constraints $(p_1, \mathsf{Smoker}(x))$ and $(p_2, \mathsf{Smoker}(x) \wedge \mathsf{Researcher}(x))$ are inconsistent when $p_1 < p_2$ because in any probability distribution, $\mathbf{P}(\mathsf{Smoker}(x)) \geq \mathbf{P}(\mathsf{Smoker}(x) \wedge \mathsf{Researcher}(x))$. In contrast, a collection of soft constraints with non-zero finite weights is always consistent. For example the MLN $\{(w_1, \mathsf{Smoker}(x)), (w_2, \mathsf{Smoker}(x) \wedge \mathsf{Researcher}(x))\}$ is consistent: the world $\{\mathsf{Smoker}(\mathtt{David})\}$ has weight w_1, while the world $\{\mathsf{Smoker}(\mathtt{David}), \mathsf{Researcher}(\mathtt{David})\}$ has weight $w_1 w_2$.

MLNs have been applied to several machine learning tasks. Applications of MLNs described in the literature typically require only a dozen constraints or less (see references in Domingos and Lowd [2009]). WebKB webpage classification models consist of a dozen or so templates [Lowd and Domingos, 2007, Mihalkova and Mooney, 2007], which during learning are instantiated to around a thousand constraints: a template is instantiated by substituting some of its variables with constants, and by associating a different weight with each instantiation. Large knowledge bases [Shin et al., 2015] require significantly more constraints. At the other extreme, the Sherlock system [Schoenmackers et al., 2010] learns over 30,000 soft Horn clause constraints using Web extractions (also see Chen and Wang [2014]).

2.6 From Soft Constraints to Independence

As outlined in §2.4.2, any query on a constrained probabilistic database $\mathbf{C}$ can be converted into a pair of queries on an unconstrained probabilistic database $\mathbf{D}$, because $\mathbf{E}_{\mathbf{C}}[Q] = \mathbf{E}_{\mathbf{D}}[Q \cdot \Delta]/\mathbf{P}_{\mathbf{D}}(\Delta)$. Moreover, we will show that any Markov Logic Network, or, more generally, any weighted database with soft constraints, can be converted into a constrained tuple-independent database. Together, these conversions form a reduction from MLNs and soft constraints, into the basic tuple-independent model. This process is depicted in Figure 2.6.

Briefly, probabilities defined by the MLN are equal to the conditional probability over a tuple-independent database. We first illustrate this reduction on a simple example, and then give the formal translation.

2.6.1 Example Reduction

Consider the database with a soft constraint in Figure 2.7. Recall that it has a single soft constraint $(v, \Phi(x, y))$, where:

$$\Phi(x, y) = (\mathsf{Manager}(x, y) \Rightarrow \mathsf{Smoker}(x))$$

We write $\mathbf{W}_M$ and $\mathbf{P}_M$ for the weight and probability distribution defined by this probabilistic database with the soft MLN constraint.

We extend the two probabilistic relations in Figure 2.7 with a third weighted relation, $\mathsf{A}(x, y)$, where all 25 ground tuples have weight v. The three relations $\mathsf{Manager}, \mathsf{Smoker}, \mathsf{A}$, without any constraints, define a tuple-independent weighted database $\mathbf{D}$. We write $\mathbf{W}_\mathbf{D}$ and $\mathbf{P}_\mathbf{D}$ for the associated weight and probability. Thus, $\mathbf{W}_\mathbf{D}(\omega)$ is the product of the weights of all tuples in ω.

Consider the following hard constraint:

$$\begin{aligned}\Delta &= \forall x \forall y(\mathsf{A}(x, y) \Leftrightarrow \Phi(x, y)) \\ &= \forall x \forall y(\mathsf{A}(x, y) \Leftrightarrow (\neg\mathsf{Manager}(x, y) \vee \mathsf{Smoker}(x))) \end{aligned} \tag{2.10}$$

Intuitively, Δ asserts that A is the set of pairs (i, j) for which the grounding $\Phi(i, j)$ holds. Let ω be any world of the MLN, and ω' be its extension with $A \stackrel{\text{def}}{=} \{(i, j) \mid \neg\mathsf{Manager}^\omega(i, j) \vee \mathsf{Smoker}^\omega(i)\}$. In other words, ω' is the unique extension of ω for which Δ holds. Then, the following is easy to check:

$$\mathbf{W}_M(\omega) = \mathbf{W}_\mathbf{D}(\omega')$$

Indeed, the weight factors in $\mathbf{W}_M(\omega)$ that correspond to grounded MLN constraints become weight factors in $\mathbf{W}_\mathbf{D}(\omega')$ that correspond to tuples in A. Moreover, the constrained probabilistic database $\mathbf{C} = (\mathbf{D}, \Delta)$ assigns zero probability to all worlds ω' of the larger schema that are not the extensions of a world ω in the MLN. This implies that, for any Boolean query Q,

$$\mathbf{P}_M(Q) = \mathbf{P}_\mathbf{C}(Q) = \mathbf{P}_\mathbf{D}(Q|\Delta). \tag{2.11}$$

In other words, the probability defined by the MLN coincides with the conditional probability in a tuple-independent database. To summarize, we compute $\mathbf{P}_M(Q)$ by first constructing a new probabilistic relation $\mathsf{A}(x, y)$, populating it with all grounded tuples $\mathsf{A}(i, j)$ over the given domain, setting their weights to v, then computing the conditional probability $\mathbf{P}(Q|\Delta)$ in the tuple-independent database. The latter, of course, can be expressed as a ratio $\mathbf{P}(Q \wedge \Delta)/\mathbf{P}(\Delta)$.

This simple technique was discussed in Van den Broeck et al. [2011]; however it has the disadvantage that the hard constraint Δ is an equivalence statement, which is difficult to handle by the lifted inference techniques that we discuss in Chapter 4. A more efficient conversion from MLNs to probabilistic database was first introduced by Jha and Suciu [2012], and simplifies the hard constraint Δ. We illustrate it on our example. As before, add a new relational symbol $\mathsf{A}(x, y)$, and set the weights of all grounded tuples $\mathsf{A}(i, j)$ to $v - 1$. Then, define the hard constraint:

$$\Delta = \forall x \forall y (\neg \mathsf{A}(x, y) \vee \neg \mathsf{Manager}(x, y) \vee \mathsf{Smoker}(x)) \tag{2.12}$$

Then Equation 2.11 continues to hold. In other words we have replaced the double implication $\mathsf{A}(x, y) \Leftrightarrow (\neg \mathsf{Manager}(x, y) \vee \mathsf{Smoker}(x))$ with a one-sided implication $\mathsf{A}(x, y) \Rightarrow (\neg \mathsf{Manager}(x, y) \vee \mathsf{Smoker}(x))$. To see why Equation 2.11 still holds, consider a world ω for the MLN, and consider one grounding of our constraint: $\delta = (\neg \mathsf{Manager}(i, j) \vee \mathsf{Smoker}(i))$. There are two cases. First, if ω satisfies δ, then δ contributes a factor v to $\mathbf{W}_M(\omega)$. On the other hand, Δ imposes no constraint on whether A contains or does not contain (i, j), thus, there are two extensions of ω that satisfy Δ: one that does not include (i, j), and the other that does include (i, j): the sum of their weights is $1 + (v - 1) = v$. Second, if ω does not satisfy δ, then δ contributes the factor 1 to $\mathbf{W}_M(\omega)$; on the other hand, Δ implies $(i, j) \notin A$, so the only possible extension is to not include (i, j), which means that, in the probabilistic database, the tuple (i, j) also contributes a factor of 1.

2.6.2 General Reduction

We now describe the general translation of a weighted database $\mathbf{D}$ with soft constraints M to a tuple-independent weighted database. First, create a new relations $\mathsf{A}_i(\mathbf{x}_i)$ for each soft constraint $(w_i, \Phi(\mathbf{x}_i))$ in M; populate A_i with all grounded tuples in the given domain, and assign to each of them the weight $w_i - 1$. Finally, define:

$$\Delta = \bigwedge_i \forall \mathbf{x}_i (\neg \mathsf{A}_i(\mathbf{x}_i) \vee \Phi_i(\mathbf{x}_i)) \tag{2.13}$$

Then, for any query Q, we have $\mathbf{P}_M(Q) = \mathbf{P}(Q|\Delta)$, where the latter probability is in a tuple-independent probabilistic database.

Discussion

We end our treatment of soft constraints with two observations. Recall that a clause is a disjunction of literals, $L_1 \vee L_2 \vee \ldots$, where each literal is either a positive relational atom $\mathsf{R}(x_1, x_2, \ldots)$ or a negated relational atom $\neg\mathsf{R}(x_1, x_2, \ldots)$. Our first observation is that, if the soft constraint $\Phi(\mathbf{x})$ is a clause, then its corresponding sentence in the hard constraint Δ is also a clause $\neg\mathsf{A}(\mathbf{x}) \vee \Phi(\mathbf{x})$. In many applications of MLNs, the soft constraint are Horn clauses, $B_1 \wedge B_2 \wedge \ldots \Rightarrow C$, and in that case the hard constraint Δ is a Horn clause as well: $A \wedge B_1 \wedge B_2 \wedge \ldots \Rightarrow C$. In other words, the formula for the hard constraint is no more complex than the original MLN constraint.

Second, the weight of the ground tuples in each new relation A_i is $w_i - 1$. When $w_i < 1$, then the tuples in A_i have a negative weight, which, in turn, corresponds to a probability that is either < 0 or > 1. This is inconsistent with the traditional definition of a probability, and requires a discussion. One simple way to try to avoid this is to replace every soft constraint (w_i, Φ_i) where $w_i < 1$ with $(1/w_i, \neg\Phi_i)$, but the new constraint $\neg\Phi_i$ is in general more complex than Φ_i; for example if Φ_i is a clause, then $\neg\Phi_i$ is a conjunctive term. However, in many applications we do not need to avoid negative weights. The marginal probability of any event Q over the tuple-independent probabilistic database is still well defined, even if some tuple probabilities are < 0 or > 1. All exact probabilistic inference methods work unchanged

in this case. Approximate methods, however, are affected, for example, we can no longer sample tuples from the relation A_i. However, collapsed sampling continues to work as long as we avoid sampling from the relations with negative weights. Finally, we notice that, certain lifted inference methods (discussed in Chapter 4) work best on formulas without negations. If all soft constraints in the MLN are positive formulas, then we can ensure that the hard constraint Δ is also a positive statement by replacing $\neg\mathsf{A}_i$ with A_i and setting the weight of A_i to $1/(w_i - 1)$ instead of $w_i - 1$.

2.7 Related Data Models

Numerous models have been proposed to represent relational uncertainty. We give a brief overview of several such models, in databases, probabilistic graphical models, as well as alternative query types.

2.7.1 Database Models

We review incomplete databases, models of other distributions and semi-structured data, and open-world probabilistic databases.

Incomplete Databases An elegant and powerful framework for representing uncertainty in databases are incomplete databases, introduced by Imielinski and Lipski [1984]. An incomplete database is defined to be a set of possible worlds; this is like a probabilistic database, but without probabilities. Thus, in an incomplete database the database can be in one of several states, and there is no preference given to one state over the other. One can define a probabilistic database as an incomplete database plus a probability distribution: for an excellent discussion of their connection we refer to Green and Tannen [2006]. Three representation formalisms have been proposed for incomplete databases: Codd-tables, v-tables, and c-tables. A v-table is a relation where the attribute values can be either constants or variables: the incomplete database is obtained by substituting each variable, independently, with a value from the domain. A Codd-table is a v-table where all variables are distinct. A c-table is a v-table where

each tuple is annotated with a condition, which is a Boolean combination of equalities involving variables and constants. Standard relations with NULL values are Codd-tables where each NULL is a distinct variable; by analogy, the variables in a v-table are sometimes called *marked nulls*. It was recently shown that SQL's semantics interprets NULLs incorrectly [Libkin, 2015].

Other Distributions and Semi-Structured Data Several applications of probabilistic databases require storing and manipulating continuous random variables. This can be achieved by allowing each attribute to store the parameters of a PDF (probability density function), from an existing library, for example single or multi-valued Gaussian distributions [Deshpande et al., 2004, Singh et al., 2008]. For example, `Temperature` may be a measured quantity and represented as a Gaussian by specifying the mean and the standard deviation. A query may check if the temperature is greater than or less than a certain value, or compare temperatures from two measurements.

There is an extensive literature on probabilistic extensions of semi-structured databases. This includes work on probabilistic XML [Nierman and Jagadish, 2002, Hung et al., 2003, Senellart and Abiteboul, 2007, Kimelfeld and Sagiv, 2007] and probabilistic RDF [Udrea et al., 2006]. Fuzzy databases [Petry, 2012] capture imprecise data through fuzzy sets instead of probability distributions.

Open-World Probabilistic Databases Our probabilistic database semantics states that all tuples in the database are possibly true, thereby making an open-world assumption [Reiter, 1978]. Nevertheless, it states that all other tuples in Tup that are missing from the database must have zero probability (cf. Equation 2.2). They may never occur in any possible world, which is still a closed-world assumption. Ceylan et al. [2016] relax this assumption by stating that missing tuples have an unknown (low) probability bounded above by a known constant. This open-world assumption matches the way information extraction systems build probabilistic knowledge bases: only some facts have been extracted from the text corpus, and all other

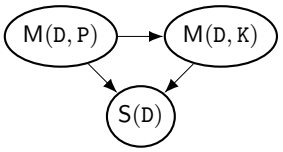
facts have an unknown probability. It also alleviates the need to store every tuple encountered during extraction. This modified assumption turns the probabilistic data model into a credal one [Cozman, 2000] based on interval probabilities [Halpern, 2003].

2.7.2 Models of Probabilistic Dependencies

Most of the work on probabilistic databases is restricted to simple distributions where the tuples are independent, or disjoint-independent. This simplifies the query semantics, because it reduces it to weighted model counting (see Chapter 3). In contrast, statistical models considered in artificial intelligence and machine learning (e.g., MLNs) have more expressive power and can directly represent richer distributions that capture dependencies between the tuples. It has been suggested that this is a limitation of probabilistic databases [Russell, 2015]. The reduction in §2.6 showed that this is not the case for MLNs, in the sense that tuple-independent probabilistic database queries can capture dependencies by conditioning on constraints. Thus, the simple data model used in probabilistic databases is not a restriction at all, but separates the specification of the uncertainties in the data from the correlations in the data, the latter being captured through constraints. See also the discussion in Suciu et al. [2011] on the design of probabilistic databases and the connection to database normalization.

Next, we further illustrate this ability by encoding a Bayesian network into a tuple-independent database. Afterwards, we briefly review complex probabilistic data models that go beyond MLNs.

Bayesian Networks and Graphical Models Probabilistic graphical models, such as Bayesian networks [Pearl, 1988, Darwiche, 2009], are the standard probabilistic model for non-relational data. Figure 2.8 depicts a simple Bayesian network over three dependent random variables. Even though this representation appears distant from tuple-independent probabilistic databases, any discrete Bayesian networks is easily reduced to a conditional probability query on a tuple-independent database. The first step of this reduction is shown in Figure 2.9, where the distribution of interest is captured by a set of soft



(a) Bayes net structure

$$\mathbf{P}(\mathsf{M}(\mathtt{D},\mathtt{P})) = 0.4$$

$$\mathbf{P}(\mathsf{M}(\mathtt{D},\mathtt{K})|\mathsf{M}(\mathtt{D},\mathtt{P})) = \begin{cases} 0.1 \text{ if } \mathsf{M}(\mathtt{D},\mathtt{P}) \\ 0.3 \text{ if } \neg\mathsf{M}(\mathtt{D},\mathtt{P}) \end{cases}$$

$$\mathbf{P}(\mathsf{S}(\mathtt{D})|\mathsf{M}(\mathtt{D},\mathtt{P}),\mathsf{M}(\mathtt{D},\mathtt{K})) = \begin{cases} 0.8 \text{ if } \mathsf{M}(\mathtt{D},\mathtt{K}) \vee \mathsf{M}(\mathtt{D},\mathtt{P}) \\ 0.5 \text{ otherwise} \end{cases}$$

(b) Conditional probabilities

Figure 2.8: A simple Bayesian network for the dependencies between random variables Manager(David, PestBye), Manager(David, KwikEMart), and Smoker(David).

Manager

CEO	Company	w
David	PestBye	0.4
David	KwikEMart	1

Smoker

Person	w
David	1

$$\begin{aligned} M = \{&(0.1, \mathsf{M}(\mathtt{D},\mathtt{K}) \wedge \mathsf{M}(\mathtt{D},\mathtt{P})), (0.9, \neg\mathsf{M}(\mathtt{D},\mathtt{K}) \wedge \mathsf{M}(\mathtt{D},\mathtt{P})), \\ &(0.3, \mathsf{M}(\mathtt{D},\mathtt{K}) \wedge \neg\mathsf{M}(\mathtt{D},\mathtt{P})), (0.7, \neg\mathsf{M}(\mathtt{D},\mathtt{K}) \wedge \neg\mathsf{M}(\mathtt{D},\mathtt{P})), \\ &(0.8, \mathsf{S}(\mathtt{D}) \wedge [\mathsf{M}(\mathtt{D},\mathtt{K}) \vee \mathsf{M}(\mathtt{D},\mathtt{P})]), (0.2, \neg\mathsf{S}(\mathtt{D}) \wedge [\mathsf{M}(\mathtt{D},\mathtt{K}) \vee \mathsf{M}(\mathtt{D},\mathtt{P})]), \\ &(0.5, \neg[\mathsf{M}(\mathtt{D},\mathtt{K}) \vee \mathsf{M}(\mathtt{D},\mathtt{P})])\}. \end{aligned}$$

Figure 2.9: A tuple-independent weighted database with soft constraints. The represented distribution is equivalent to the Bayesian network distribution in Figure 2.8.

constraints (i.e., an MLN) on top of a weighted tuple-independent database. By further applying the reduction from soft constraints to conditional queries in §2.6, we can effectively capture the dependencies in the Bayesian network within the tuple-independent data model. This type of reduction, from graphical models to a constraint on independent variables, is known as a weighted model counting encoding in graphical models [Chavira and Darwiche, 2005, 2008, Sang et al., 2004]. The computational aspects of weighted model counting will be the topic of Chapter 3. It naturally exploits structure in the distribution, such as equal parameters and determinism, and attains state-of-the-art performance [Darwiche et al., 2008, Choi et al., 2013].

Statistical Relational Models and Probabilistic Programming The machine learning and knowledge representation communities have developed models that are very related to probabilistic databases. The goal of these *statistical relational models* [Getoor and Taskar, 2007] is to have a concise high-level description of a large graphical models with repeated structure. Given a relational database containing the known tuples, the statistical relational model induces a classical graphical model over the uncertain tuples. Just like probabilistic databases, the possible worlds of these models are classical relational databases. A large number of such template languages has been proposed, including plate models [Buntine, 1994], RBNs [Jaeger, 1997], PRMs [Friedman et al., 1999, Getoor et al., 2001], BLPs [Kersting and De Raedt, 2001], parfactors [Poole, 2003], LBNs [Fierens et al., 2005], PSL [Kimmig et al., 2012], and MLNs; collectively referred to as the alphabet soup of statistical relational learning.

A related line of research in artificial intelligence seeks to extend first-order logic and logic programming with probabilities. Based on the seminal work of Nilsson [1986], Halpern [1990], and Bacchus [1991], a rich collection of *probabilistic logic programming* languages was developed, including PHA/ICL [Poole, 1993, 1997], SLPs [Muggleton, 1996], PRISM [Sato and Kameya, 1997], ProbLog [De Raedt et al., 2007], ProPPR [Wang et al., 2013], and GDatalog [Barany et al., 2016]. Most of these languages are based on the distribution semantics [Sato,

1995]. In fact, many can be seen as executing datalog or logic program queries on a tuple-independent probabilistic database following the semantics of §2.3. Moreover, the LPAD/CP-Logic languages also support disjoint-independent sets of tuples [Vennekens et al., 2004, 2009].

More recently, probabilistic programming languages have taken up the role of representing complex distributions over structured data [Milch et al., 2007, Goodman et al., 2008, Pfeffer, 2009, McCallum et al., 2009]. Probabilistic logic programming languages are one popular approach; others extend functional or even imperative languages with probabilistic constructs. Several proposals aim to augment description logics with probabilities [Heinsohn, 1994, Lukasiewicz, 2008]. There has also been substantial interest in continuous-variable extensions of probabilistic logic programs [Nitti et al., 2016].

Intricate connections exist between all these languages. For example, MLNs can be regarded as a maximum-entropy probabilistic logic [Kern-Isberner and Lukasiewicz, 2004] in the tradition of Nilsson [Paskin, 2002]. Some probabilistic logic programs can reduce to simpler queries on tuple-independent probabilistic databases [Van den Broeck et al., 2014]. These are beyond the scope of this survey.

2.7.3 Alternative Query Semantics

Two classes of queries go beyond the expectation semantics of §2.3.

Meta-Queries The probabilistic database system Trio [Benjelloun et al., 2006b] supports queries where the probability can be tested and manipulated explicitly. For example, in Trio we could ask the following query on the relation Researcher in Figure 2.2: *retrieve all affiliations that have expertise in both databases and vision with probability* > 0.8. This query is a departure from our definition of a query as a random variable over the possible worlds, since the query has no meaning over a single world, only over the entire probability distribution. Similar queries are implemented in ProbLog, and called Meta-Calls [De Raedt and Kimmig, 2015]. The formal semantics of such queries is more complex than for the queries considered in this survey, we refer the reader to Fagin et al. [1990] and also to the discussion by Moore et al. [2009].

Other variations to the query semantics have been considered in the literature. Li and Deshpande [2009] propose *consensus answers*, which are answers that minimize the distance to all possible answers, i.e. they are a consensus over the possible answers. Sensitivity analysis for query answers was discussed by Kanagal et al. [2011].

Top-k Queries Recall that a set-valued query returns a set of answers, and are by far the most common SQL queries in practice. If the database is probabilistic, each of these answers is uncertain, and it is reasonable to require that the answers be sorted in decreasing order of their confidence and, moreover, restrict the answers to just the top k, where k is, say, $k \approx 10 \cdots 20$. This opens up the possibility for optimizations, since the system does not have to compute the probabilities all answers that are not in the top k. Ré et al. [2007] describe an approach to improve the query performance by focusing the probabilistic inference to just the top k answers. It uses as a black box any approximation algorithm for computing the output probability, and runs this algorithm one step at a time on each candidate output tuple, and drops an tuple from the list of candidates when its current confidence interval is strictly dominated by current confidence intervals of at least k other tuples.

A different top-k problem studies queries that have a particular score attribute that the system has to use to sort the answers. For example, the query may retrieve a list of hotels, and order them in increasing order of the price; or in decreasing order of their average review score; or by some other deterministic criterion. When the data is deterministic, then it is clear how to sort the output, namely by the score. When the data is probabilistic we need a sorting criterion that accounts for both the score and the confidence. For example, should a hotel with very good reviews but low confidence be listed before, or after a hotel with average reviews but much higher confidence? Several semantics have been discussed in the literature, we refer the reader to Zhang and Chomicki [2008] for a survey and critique. Li et al. [2011] proposed a solution based on learning ranking function based on user inputs.

3

Weighted Model Counting

Model counting, and its generalization to weighted model counting (WMC), are some of the most fundamental problems in computer science. Several open-source tools exists for these tasks. Our interest in them stems from the fact that evaluation of Boolean queries on a probabilistic databases can be reduced to WMC, and, therefore, tools and theory for WMC can be deployed to perform and analyze query evaluation. In this chapter we give a brief background on WMC and describe its connection to query evaluation on probabilistic databases.

3.1 Three Variants of Model Counting

Let F be a Boolean formula over variables from the set $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$. A *model for* F is a satisfying assignment, in other words a function $\theta : \mathbf{X} \to \{0, 1\}$ s.t. F evaluates to true: $\theta(F) = 1$. With some abuse, we will interchangeably use $0/1$ and `false`/`true`. While we find it convenient to write $\theta(F) = 1$ for satisfaction of Boolean formulas, note that it is equivalent to writing $\theta \models F$, seeing θ as a possible world. A *monotone* Boolean formula is one that can be written without using the negation symbol. It is easily satisfied by the assignment that sets all variables to `true`.

X_1	X_2	X_3	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Figure 3.1: Truth table for the formula $F = (X_1 \vee X_2) \wedge (X_1 \vee X_3) \wedge (X_2 \vee X_3)$.

Problem 1: Model Counting

The number of models of F is denoted $\#F = |\{\theta \mid \theta(F) = 1\}|$, and the model counting problem is *given a Boolean formula F, compute $\#F$*. For example, if $F = (X_1 \vee X_2) \wedge (X_1 \vee X_3) \wedge (X_2 \vee X_3)$ then $\#F = 4$, as is clear from its truth table in Figure 3.1.

In general, the model counting problem is computationally hard. An easy way to see this is to observe that any oracle for computing $\#F$ can be used to check if F is satisfiable, by simply testing if $\#F \geq 1$, and since the satisfiability problem (SAT) is NP-hard, the counting problem must also be hard. Recall that the class NP is the class of decision problems that can be solved by a polynomial time non-deterministic Turing machine; SAT is NP-complete. The model counting problem is not a decision problem, and therefore is not in NP. The model counting problem belongs to the class #P, defined as the class of functions f for which there exists a polynomial time non-deterministic Turing Machine such that, for any input x, the number of accepting computations of the Turing Machine is $f(x)$. Valiant [1979b,a] introduced the class #P and proved that the model counting problem is #P-complete.

Rather surprisingly, even if one restricts the class of Boolean formulas to apparently simple formulas, the model counting problem remains #P-hard. A particular class that is of special interest to probabilistic databases is that of *Positive Partitioned 2CNF* (PP2CNF) formulas. A PP2CNF is a Boolean formula of the form $F = \bigwedge_{(i,j)\in E}(X_i \vee Y_j)$,

where $\mathbf{X} = \{X_1, \ldots, X_n\}$, $\mathbf{Y} = \{Y_1, \ldots, Y_n\}$ are two disjoint sets of variables, and $E \subseteq [n] \times [n]$. The satisfiability problem for PP2CNF is trivial, because any PP2CNF formula is trivially satisfied by setting all variables to true. Therefore, it is quite surprising that Provan and Ball [1983] proved that model counting is hard:

Theorem 3.1 (PP2CNF)**.** The model counting problem for PP2CNF formulas is #P-hard.

The *dual* of a Boolean formula $F(X_1, \ldots, X_n)$ is defined as $F^* = \neg F(\neg X_1, \ldots, \neg X_n)$. Equivalently, it is obtained from F by replacing $\wedge, \vee$ with $\vee, \wedge$ respectively. The model counting problems for F and F^* are equivalent, since $\#F = 2^n - \#F^*$ and, therefore, the result by Provan and Ball [1983] immediately applies to PP2DNF formulas, which are formulas of the form $\bigvee_{(i,j)\in E} X_i \wedge Y_j$.

Problem 2: Probability Computation

In probabilistic databases we are interested in computing the probability of a Boolean formula, which turns out to be related to model counting. Let $p : \mathbf{X} \to [0, 1]$ be a probability function. Equivalently, we denote it as a sequence $(p_i)_{i=1,n}$, where, for each $i = 1, n$, the number $p_i \in [0, 1]$ denotes $p(X_i)$. We define a probability space whose set of outcomes is the set of assignments θ, as follows. For each variable X_i, set randomly $\theta(X_i) = 1$ with probability p_i, or $\theta(X_i) = 0$ with probability $1 - p_i$. Repeated this independently for each variable X_i, $i = 1, n$. This defines a probability space on the set of assignments θ. Concretely, the probability $\mathbf{P}(\theta)$ of an assignment θ, and the marginal probability $\mathbf{P}(F)$ of a Boolean formula F are given by:

$$\mathbf{P}(\theta) = \prod_{i:\theta(X_i)=0} (1 - p_i) \times \prod_{i:\theta(X_i)=1} p_i \qquad \mathbf{P}(F) = \sum_{\theta:\theta(F)=1} \mathbf{P}(\theta)$$

The *probability computation problem* is: given a Boolean formula F and rational numbers $p_i \in [0, 1]$, $i = 1, n$, compute $\mathbf{P}(F)$. In the special case when $p_1 = \cdots = p_n = 1/2$, the probability is $\mathbf{P}(F) = \#F/2^n$, and therefore the probability computation problem generalizes model counting, and is at least as hard.

Expansion and Independence Properties

We review some simple properties of the probability of a Boolean formula. *Shannon's expansion* formula is:

$$\mathbf{P}(F) = (1 - p(X)) \cdot \mathbf{P}(F[X = 0]) + p(X) \cdot \mathbf{P}(F[X = 1])$$

where X is any variable, and $F[X = v]$ replaces X in F by value v. By repeatedly applying the Shannon expansion one can compute $\mathbf{P}(F)$ in time $\leq 2^n$. A more efficient way is to apply the independence rule.

Lemma 3.2. If F_1, F_2 do not share any common variables ($\texttt{Vars}(F_1) \cap \texttt{Vars}(F_2) = \emptyset$) then they are independent probabilistic events, and

$$\mathbf{P}(F_1 \wedge F_2) = \mathbf{P}(F_1) \cdot \mathbf{P}(F_2)$$
$$\mathbf{P}(F_1 \vee F_2) = 1 - (1 - \mathbf{P}(F_1)) \cdot (1 - \mathbf{P}(F_2)).$$

One has to be careful when reasoning about independence: the converse, namely that the equalities above imply the absence of a common variable, fails in general, and only holds under certain conditions. The lifted inference techniques discussed in Chapter 4 will require necessary and sufficient criteria for independence, which is why we are interested in precisely characterizing it. In general, F_1, F_2 may share some common variables, and at the same time be independent probabilistic events for particular values of the probabilities p_i. A concrete example[1] is $F_1 = (X \vee Y) \wedge (\neg X \vee Z)$ and $F_2 = (X \vee U) \wedge (\neg X \vee W)$, where all Boolean variables have probability $1/2$; then $\mathbf{P}(F_1 \wedge F_2) = \mathbf{P}(F_1)\mathbf{P}(F_2) = 1/2$. Thus, we have an example of two formulas F_1, F_2 that have a common variable X, and are also independent. Therefore, in general, independence does not imply disjoint sets of variables.

On the other hand, if F_1, F_2 are independent for *every* choice of probabilities $p_i \in [0, 1]$, $i = 1, n$, then F_1, F_2 have no common variables; this provides a weak converse to Lemma 3.2. Indeed, if they shared at least one common variable X, then $\mathbf{P}(F_1 \wedge F_2)$ is a linear

[1] We invite the reader to verify the following more general statement. If F_1, F_2 share a single common variable X, and $\mathbf{P}(F_1[X = 0]) = \mathbf{P}(F_1[X = 1])$, and $\mathbf{P}(F_2[X = 0]) = \mathbf{P}(F_2[X = 1])$, then $\mathbf{P}(F_1 \wedge F_2) = \mathbf{P}(F_1)\mathbf{P}(F_2)$. To check, it suffices to apply Shannon expansion on X.

polynomial in $p = p(X)$, while $\mathbf{P}(F_1)\mathbf{P}(F_2)$ is a polynomial of degree 2 in p, and they cannot be identical for all values of $p \in [0, 1]$.

Finally, if F_1, F_2 are monotone Boolean formulas then the converse of Lemma 3.2 holds in the following sense: if there exist probability values $p_i \in (0, 1)$ such that $\mathbf{P}(F_1 \wedge F_2) = \mathbf{P}(F_1)\mathbf{P}(F_2)$ then F_1, F_2 do not share any common variables, see Miklau and Suciu [2007].

Problem 3: Weighted Model Counting

The third problem discussed here is a variant of the probability computation problem that is closer in spirit to model counting. Let $w : \mathbf{X} \to \mathbb{R}$ be a function that associates a weight $w(X)$ to each variable X. As with probabilities, we denote $w_i = w(X_i)$ and use the sequence $(w_i)_{i=1}^n$ to present the function w. The weight of an assignment $w(\theta)$ is the product of the weights of the variables set to true,[2] and the weighted model count of a Boolean formula, WMC(F), is the sum of weights of its models, formally:

$$w(\theta) = \prod_{i:\theta(X_i)=1} w_i \qquad \text{WMC}(F) = \sum_{\theta:\theta(F)=1} w(\theta)$$

The *Weighted Model Counting (WMC) Problem* is the following: *given a Boolean formula F and weights w_i for the Boolean variables X_i, compute WMC(F)*. In the special case when $w_1 = \cdots = w_n = 1$ then WMC$(F) = \#F$, and therefore the WMC problem is also a generalization of the model counting problem. In fact, the WMC problem is equivalent to the probability computation problem, in the following sense. If we set every weight w_i to be the odds of the probability p_i, $w_i \stackrel{\text{def}}{=} p_i/(1 - p_i)$ for all i, then $\mathbf{P}(F) = \text{WMC}(F)/Z$ where $Z \stackrel{\text{def}}{=} \prod_i(1 + w_i)$ is the normalization factor. Or, equivalently, WMC$(F) = \mathbf{P}(F)/\prod_i(1 - p_i)$.

[2] It is also common to associate weights $\bar{w}_i$ with assigning *false* to X_i (or equivalently, to associate a weight with all literals) [Chavira and Darwiche, 2008]. The weight of a model then becomes $w(\theta) = \prod_{i:\theta(X_i)=1} w_i \prod_{i:\theta(X_i)=0} \bar{w}_i$. These definitions are interchangeable by normalization as long as $w_i + \bar{w}_i \neq 0$.

3.2 Relationships Between the Three Problems

To summarize, we discussed three related problems on a Boolean formula: the model counting, the probability computation, and the weighted model counting problem. Their relationship is captured by the following:

Proposition 3.1. (1) Model counting is a special case of weighted model counting: if $w(X_i) \stackrel{\text{def}}{=} 1$ for all i, then $\text{WMC}(F) = \#F$. (2) Weighted model counting and probability computation are virtually the same: if $w_i \stackrel{\text{def}}{=} p_i/(1-p_i)$ for all i, then $\mathbf{P}(F) = \text{WMC}(F)/Z$ where $Z \stackrel{\text{def}}{=} \prod_i (1+w_i)$ is the normalization factor.

For a simple illustration, consider the Boolean formula

$$F = (X_1 \vee X_2) \wedge (X_1 \vee X_3) \wedge (X_2 \vee X_3)$$

whose truth table is shown in Figure 3.1. The following are easily checked:

$$\begin{aligned} \#F &= 4 \\ \text{WMC}(F) &= w_2w_3 + w_1w_3 + w_1w_2 + w_1w_2w_3 \\ \mathbf{P}(F) &= (1-p_1)p_2p_3 + p_1(1-p_2)p_3 + p_1p_2(1-p_3) + p_1p_2p_3 \end{aligned}$$

Notice that we did not state that the probability computation problem is equivalent to weighted model counting, and this raises a natural question: is the probability computation problem harder than model counting? We end this section by answering this question: the short answer is both yes and no.

On the one hand, there exist classes of Boolean formulas for which the model counting problem is in PTIME while the probability computation problem is #P-hard. A consequence of this subtle distinction is explained in §4.6, where we show that some queries can be evaluated in PTIME over symmetric databases, where many p_i are identical, but their complexity is #P-hard over general probabilistic databases. For example, consider the family of formulas $F_n \stackrel{\text{def}}{=} \bigwedge_{i,j\in[n]}(X_i \vee Z_{ij} \vee Y_j)$: thus, for every n there is a single formula with n^2+2n Boolean variables. We show that computing $\#F_n$ is easy, yet $\mathbf{P}(F_n)$ is #P-hard. To

compute $\#F_n$, suppose we assign k variables $X_i = 0$ and ℓ variables $Y_j = 0$, and assign the other variables X_i, Y_j to 1. Then $k\ell$ variables Z_{ij} must be set to 1, while the other $n^2 - k\ell$ variables Z_{ij} can be either 0 or 1; therefore $\#F_n = \sum_{k,\ell} \binom{n}{k}\binom{n}{\ell} 2^{n^2-k\ell}$ and can obviously be computed in PTIME. On the other hand, if we are allowed to set the probabilities of the variables freely, then computing $\mathbf{P}(F_n)$ is #P-hard, by reduction from a PP2CNF formula: set $p(X_i) = p(Y_j) = 1/2$ forall i, j, and set $p(Z_{ij}) = 0$ for $(i, j) \in E$, and $p(Z_{ij}) = 1$ for $(i, j) \notin E$.

On the other hand, for any Boolean formula F and rational numbers $p_i \in [0, 1]$, one can reduce the probability computation problem $\mathbf{P}(F)$ to the model counting problem $\#G$ of some other formula G, which depends both on F and the numbers p_i (or to the ratio of two model counts). A consequence is that, if we have an efficient algorithm for model counting for *all* formulas, then we can use it to compute probabilities of all formulas. We briefly describe here the reduction, adapting from Chakraborty et al. [2015]. For any m Boolean variables $Y_1, \ldots, Y_m$ and number $k \leq 2^m$, we define $C_{<k}$ the Boolean formula asserting that the number whose binary representation is $Y_m Y_{m-1} \cdots Y_1$ is $< k$: $C_{<k} \stackrel{\text{def}}{=} \neg Y_m \text{op}_m(\neg Y_{m-1} \text{op}_{m-1} \cdots (\neg Y_1 \text{op}_1 1) \cdots)$, where op_j is $\wedge$ or $\vee$, depending on whether the i'th bit of $k - 1$ is 0 or 1 respectively. For example, if $k = 6$ then 5 in binary is 0101 and $C_{<6} = \neg Y_4 \wedge (\neg Y_3 \vee (\neg Y_2 \wedge (\neg Y_1 \vee 1)))$ asserts that $Y_4 Y_3 Y_2 Y_1 < 6$. Let F be a Boolean formula over variables X_i, $i = 1, n$, and let $0 < p_i < 1$ be their probabilities given as rational numbers. Assume first that all denominators are powers of 2, $p_i = k_i/2^{m_i}$. Create m_i fresh variables $Y_{i1}, Y_{i2}, \ldots, Y_{im_i}$, substitute in F every variable X_i with $C_{<k_i}$, and denote G the resulting formula. If we set the probability of each Boolean variable Y_{ij} to $1/2$, then $\mathbf{P}(C_{<k_i}) = k_i/2^{m_i} = p_i$, and one can check that $\mathbf{P}(F) = \mathbf{P}(G)$, and the latter is $\#G/2^m$ where $m = \sum m_i$ is the total number of variables in G. If the denominators of p_i are not powers of 2, then, assuming $p_i = k_i/n_i$, let m_i be such that $n_i < 2^{m_i}$. Define G as before, obtained by substituting $C_{<k_i}$ for X_i in F. Let $C_{\geq 2^{m_i} - n_i + k_i}$ be the formula asserting that the string $Y_{im_i} \cdots Y_{i1}$ is $\geq 2^{m_i} - n_i + k_i$, and denote $H = \bigwedge_i (C_{<k_i} \vee C_{\geq k_i})$. Intuitively, H restricts the code $Y_{im_i} \cdots Y_{i1}$ to be either $< k_i$ or $\geq 2^{m_i} - n_i + k_i$, and therefore $\mathbf{P}(C_{<k_i}|H) = p_i$, $\mathbf{P}(\neg C_{<k_i}|H) = 1 - p_i$, implying that $\mathbf{P}(F) = \mathbf{P}(G|H)$. The latter is equal to $\#(G \wedge H)/\#H$, proving the claim.

3.3 First-Order Model Counting

Fix a first-order sentence Q and a finite domain D. By expanding Q over the domain D we obtain a Boolean formula, called the *lineage*, or *provenance*, or *grounding* of Q. The first-order model counting problem is the model counting problem for the grounded Boolean formula; similarly one can consider the probability computation problem, and the weighted model counting problem, for the grounded Boolean formula obtained from Q and D. In probabilistic databases, the domain is the active domain of a probabilistic database $\mathbf{D}$, the query is the user's query, and the probabilities are defined by $\mathbf{D}$. In this section we review the First-Order Model Counting problem, and its variations to probability computation and weighted model counting.

Recall that $\text{Tup}(D)$ denotes the set of grounded atoms over the domain D. We view each grounded tuple as a Boolean variable, and consider Boolean formulas over the variables $\text{Tup}(D)$, for example $R(a,b) \wedge S(b,c) \vee R(a,c) \wedge \neg S(c,a)$.

Definition 3.1. Given first-order sentence Q and a domain D, the *lineage* or *grounding* of Q over D is a Boolean formula denoted $F_{Q,D}$ defined inductively on the structure of Q:

- If Q is a grounded atom $R(a,b,\ldots)$ then $F_{Q,D} = Q$.
- If Q is a grounded equality predicate $a = a$ then $F_{Q,D} = \texttt{true}$.
- If Q is a grounded equality predicate $a = b$, for distinct constants a, b, then $F_{Q,D} = \texttt{false}$.
- If Q is $Q_1 \vee Q_2$ then $F_{Q,D} = F_{Q_1,D} \vee F_{Q_2,D}$
- If Q is $Q_1 \wedge Q_2$ then $F_{Q,D} = F_{Q_1,D} \wedge F_{Q_2,D}$
- If Q is $\neg Q_1$, then $F_{Q,D}$ is $\neg F_{Q_1,D}$.
- IF Q is $\exists x Q_1$, then $F_{Q,D} = \bigvee_{a \in D} F_{Q_1[a/x],D}$.
- IF Q is $\forall x Q_1$, then $F_{Q,D} = \bigwedge_{a \in D} F_{Q_1[a/x],D}$.

Described succinctly, the lineage is obtained by replacing $\exists$ with $\bigvee$, replacing $\forall$ with $\bigwedge$, and leaving all other Boolean operations unchanged. We abbreviate $F_{Q,D}$ with F_Q when D is understood from the context.

The definition of lineage given above is purely syntactic, in other words $F_{Q,D}$ is a Boolean *expression*, of size is $O(|D|^k)$ where k is the number of logical variables occurring in the sentence Q. This means that, for a fixed Q, the size of the lineage is polynomial in the size of the domain.

Alternatively, we give an equivalent, semantic definition of the lineage, by defining the Boolean formula $F_{Q,D}$ on all possible valuations $\theta : \text{Tup}(D) \rightarrow \{0,1\}$. Recall that a (standard) database is a finite structure $\omega \subseteq \text{Tup}(D)$; using standard terminology in first-order logic, and in databases, we say that Q is true in ω, and write $\omega \models Q$, or $Q(\omega) = \texttt{true}$, if the sentence holds in ω. We denote the characteristic function associated to ω by $\theta_\omega : \text{Tup}(D) \rightarrow \{0,1\}$; in other words, $\theta_\omega(t) \stackrel{\text{def}}{=} 1$ if $t \in \omega$, and $\theta_\omega(t) \stackrel{\text{def}}{=} 0$ if $t \notin \omega$. The following theorem gives a semantic definition of $F_{Q,D}$, by specifying which assignments θ make it true. We invite the reader to check that this is equivalent to the syntactic definition given above.

Theorem 3.3. Fix a domain D and a first-order sentence Q. Then, for any database $\omega \subseteq \text{Tup}(D)$: $\omega \models Q$ iff $\theta_\omega(F_{Q,D}) = 1$.

Example 3.1. For a simple illustration, consider the formula:

$$Q = \forall d(\mathsf{Rain}(d) \Rightarrow \mathsf{Cloudy}(d))$$

Then the lineage over the domain $D = [7]$ is:

$$F_{Q,[7]} = (\mathsf{Rain}(1) \Rightarrow \mathsf{Cloudy}(1)) \wedge \cdots \wedge (\mathsf{Rain}(7) \Rightarrow \mathsf{Cloudy}(7))$$

Consider the database $\omega = \{\mathsf{Rain}(2), \mathsf{Cloudy}(1), \mathsf{Cloudy}(2)\}$. Then $\omega \models Q$, and it can be easily checked that $F_{Q,[7]}$ is also true under the assignment θ_ω defined by ω, namely θ_ω assigns $\mathsf{Rain}(2) = \mathsf{Cloudy}(1) = \mathsf{Cloudy}(2) = 1$ and assigns all other Boolean variables to 0.

In the *first-order model counting* problem (FOMC) we are given Q, D, and the problem is to compute the number of models $\#F_{Q,D}$.

Equivalently, this is the number of databases (or first order structures) $\omega \subseteq \text{Tup}(D)$ for which $\omega \models Q$. In the *Weighted FOMC* problem, we are given, in addition, a function $w : \texttt{Tup}(D) \to \mathsf{R}$, and the problem is to compute $\text{WMC}(F_{Q,D})$. Finally, in the *first-order probability computation* problem we are given a function $p : \texttt{Tup}(D) \to [0,1]$ and ask for $\mathbf{P}(F_{Q,D})$.

Consider now a probabilistic database $\mathbf{D}$. Recall that $\mathbf{D}$ can be seen as a deterministic database $\mathbf{T}$ and a function $p : \mathbf{T} \to [0,1]$. Extend p to all tuples, $p : \text{Tup}(D) \to [0,1]$, by setting $p(t) = 0$ forall $t \notin \mathbf{T}$. Theorem 3.3 immediately implies the following.

Proposition 3.2. The probability of the lineage of a query Q is identical to the probabilistic database query probability: $\mathbf{P}(F_{Q,D}) = \mathbf{P}_{\mathbf{D}}(Q)$.

Thus, the query evaluation problem in probabilistic databases can be reduced to computing the probability of a Boolean formula, namely the lineage of the query over the active domain of the database. We call this the *grounded* query evaluation approach; it is also called *intensional query evaluation* [Suciu et al., 2011]. An alternative approach is *lifted inference* (or *extensional query evaluation*), which we will discuss in Chapter 4. Lifted inference is much more efficient, but, as we shall see, works only for a subset of queries. All probabilistic database systems use lineage-based query evaluation, at least as a backup when lifted inference fails.

We end this section with a discussion on how to compute the lineage expression $F_{Q,D}$.

Specializing the Lineage to a Database The fundamental property of the lineage expression $F_{Q,D}$ is given by Theorem 3.3: a set of tuples ω satisfies the sentence Q iff the valuation θ_ω satisfies $F_{Q,D}$. In practice, we are always given a a concrete database $\mathbf{T}$, and the set of tuples are always a subset, $\omega \subseteq \mathbf{T}$. In that case it is redundant to use Boolean variables for tuples that are not in the database $\mathbf{T}$. Instead, we simplify $F_{Q,D}$ by substituting with `false` all Boolean variables corresponding to tuples that are not in $\mathbf{T}$, and denote $F_{Q,\mathbf{T}}$ the resulting expression. This is called the lineage of Q over the database $\mathbf{T}$. By specializing the lineage to only the tuples in $\mathbf{T}$ we can significantly reduce its size.

For a simple example, consider the relational database schema

$$\mathsf{Researcher}(\texttt{Name}, \texttt{Expertise}, \texttt{Affiliation})$$
$$\mathsf{University}(\texttt{UName}, \texttt{City})$$

The following conjunctive query checks if there exists any researcher with expertise in `Vision` affiliated with some university in Seattle:

$$Q = \exists x \exists y\, \mathsf{Researcher}(x, \texttt{Vision}, y) \wedge \mathsf{University}(y, \texttt{Seattle})$$

If the database $\mathbf{T}$ has an active domain D of size n, then there are $n^3 + n^2$ Boolean variables, one for each ground tuple $\mathsf{Researcher}(a, b, c)$ and one for each ground tuple $\mathsf{University}(c, d)$, and the lineage $F_{Q,D}$ is a DNF formula with $O(n^3)$ terms. However, if `Name`, `UName` are keys and `Affiliation` is a foreign key, then the database $\mathbf{T}$ contains only $2n$ tuples, and therefore the lineage over the database $F_{Q,\mathbf{T}}$ is much smaller. For example, it may be the following Boolean expression:

$$\begin{aligned}&\mathsf{Researcher}(\texttt{Alice}, \texttt{Vision}, \texttt{UPenn}) \wedge \mathsf{University}(\texttt{UPenn}, \texttt{Philadelphia})\\ \vee\ &\mathsf{Researcher}(\texttt{Bob}, \texttt{Vision}, \texttt{Brown}) \wedge \mathsf{University}(\texttt{Brown}, \texttt{Providence})\\ \vee\ &\mathsf{Researcher}(\texttt{Carol}, \texttt{Vision}, \texttt{UCLA}) \wedge \mathsf{University}(\texttt{UCLA}, \texttt{LosAngeles})\end{aligned}$$

Lineage of a Set-Valued Query So far we have defined the lineage only for a sentence Q, or, equivalently, for a Boolean query. We now extend the definition to a set-valued query. In this case the lineage is defined as a set of lineages, one for each possible output tuple. For example, consider the following query, returning all expertises of researchers affiliated with some university in Seattle:

$$Q(z) \text{ :- } \mathsf{Researcher}(x, z, y), \mathsf{University}(y, \texttt{Seattle}).$$

Then we have to compute the lineage separately for each value z in the active domain of the `Expertise` attribute. When $z =$ `Vision` then the lineage is given above; when $z =$ `Graphics`, or `Databases`, or some other expertise, then the lineage is a similar Boolean formula.

Computing Lineage in the Database Engine The provenance computation system Perm by Glavic and Alonso [2009] developed a practical method for pushing the provenance (lineage) computation

inside the database engine, using query rewriting. We describe here the main idea, for conjunctive queries. In this case, the conjunctive query has the from `select distinct attributes` in SQL, and it is rewritten into a query of the of the form `select * order by attributes`; each group with the same values of `attributes` represents one DNF expression, where each minterm is given by one row in the answer, and consists of the conjunction of the tuples in that row. For example, the query Q above written in SQL is:

```
select distinct r.expertise
from Researcher r, University u
where r.affiliation = u.uname
      and u.city = "Seattle"
```

The lineage is a DNF formula whose terms are given by the rows returned by the following query:

```
select *
from Researcher r, University u
where r.affiliation = u.uname
      and u.city = "Seattle"
order by r.expertise
```

The only change is replacing the `select distinct` clause with `select *` and adding an `order by` clause, so that we can easily read the provenance expressions associated to each output tuple `r.expertise`. Namely, for each fixed expertise e, the set of answers with `r.expertise` $= e$ define one DNF expression, and each row `n,e,u,c` in the answer represents a prime implicant $\mathsf{Researcher}(n, e, u) \wedge \mathsf{University}(u, c)$.

The Lineage of a Constraint When the sentence Q is a constraint rather than a Boolean query, then it has universal quantifiers and negations, which require a more careful construction of the SQL query computing the lineage. We illustrate the main idea on the following constraint, asserting that all `Vision` experts are affiliated with some university in Seattle:

$$\forall x \forall y \, (\mathsf{Researcher}(x, \texttt{Vision}, y) \Rightarrow \mathsf{University}(y, \texttt{Seattle}))$$

We first write it in CNF clausal form:

$$\forall x \forall y \, (\neg \mathsf{Researcher}(x, \texttt{Vision}, y) \vee \mathsf{University}(y, \texttt{Seattle}))$$

The lineage is a CNF expression, and we wish to design a SQL query that computes the clauses of this expression. We need to be careful, however, since the expression $\neg\mathsf{Researcher}(x, \texttt{Vision}, y) \vee \mathsf{University}(y, \texttt{Seattle})$ is domain dependent, hence we can no longer simply write it as a `select *` query. For any name n and university u, denote the following ground tuples:

$$R_{nu} \stackrel{\text{def}}{=} \mathsf{Researcher}(n, \texttt{Vision}, u)$$
$$U_u \stackrel{\text{def}}{=} \mathsf{University}(u, \texttt{Seattle})$$

The lineage over the entire active domain is $\bigwedge_{n,u \in D}(\neg R_{nu} \vee U_u)$. To specialize it to a particular database $\mathbf{T}$ we need to consider three cases:

- $R_{nu} \notin \mathbf{T}, U_u \in \mathbf{T}$, then $(\neg R_{nu} \vee U_u) \equiv \texttt{true}$, and the clause gets removed.
- $R_{nu} \in \mathbf{T}, U_u \notin \mathbf{T}$, then $(\neg R_{nu} \vee U_u) \equiv \neg R_{nu}$, and the clause becomes a unit clause.
- $R_{nu} \in \mathbf{T}, U_u \in \mathbf{T}$, then the clause remains unchanged, $\neg R_{nu} \vee U_u$.

Therefore, we need to compute two SQL queries, corresponding to cases 2 and 3 above:

```
select distinct r.Name, r.affiliation
       from Researcher r
where r.expertise = 'Vision'
  and not exists (select * from University u
                  where u.city = 'Seattle'
                    and r.affiliation = u.uname);

select * from Researcher r, University u
where u.city = 'Seattle'
  and r.expertise = 'Vision'
  and r.affiliation = u.uname
```

The answers to the first SQL query are interpreted as unit clauses $\neg R_{nu}$, and the answers to the second SQL query are interpreted as clauses with two literals $\neg R_{nu} \vee U_u$. These query rewritings do not assume any key or foreign key constraints in the database. If `UName` is a key in University and `Affiliation` is a foreign key, then the first SQL query above simplifies: it suffices to lookup the unique city of the researcher's affiliation, and check that it is not Seattle:

```
select distinct r.Name, r.affiliation
from Researcher r, University u
where u.city != 'Seattle'
  and r.expertise = 'Vision'
  and r.affiliation = u.uname
```

3.4 Algorithms and Complexity for Exact Model Counting

Consider the model counting problem: given a Boolean formula F, compute $\#F$. In this section we study the exact model counting problem; we discuss approximations in the next section. Valiant has established that model counting is #P-hard, even for 2CNF or 2DNF formulas. In this paper we are concerned with formulas that can arise as groundings of a fixed first-order (FO) sentence, in other words we are interested in first-order model counting. In that case, if we fix the FO sentence Q, the complexity can be either #P-hard or in PTIME, depending on Q.

We start by showing that weighted model counting is #P-hard even if we fix the FO sentence Q. In other words, in general, first-order model counting for a fixed Q does not become easier than model counting, even for relatively simple sentences Q.

Theorem 3.4. Consider the conjunctive query H_0

$$H_0 = \exists x \exists y\, R(x) \wedge S(x, y) \wedge T(y).$$

Then, computing $\mathbf{P}(H_0)$ over a tuple-independent probabilistic database is #P-hard in the size of the input database.

Proof. We show hardness of H_0 by reduction from PP2DNF. Given a PP2DNF instance $F = \bigvee_{(i,j)\in E} X_i \wedge Y_j$ where $E \subseteq [n] \times [n]$,

define a database $\mathbf{T}$ where $R = T = [n]$, $S = E$, and set the probabilities as follows: $p(R(i)) = p(T(i)) = 1/2$ forall $i \in [n]$, $p(S(i,j)) = 1$ forall $(i,j) \in E$. Then the lineage of H_0 on $\mathbf{T}$ is $F_{H_0,\mathbf{T}} = \bigvee_{(i,j)\in E} R(i) \wedge S(i,j) \wedge T(j)$, and its probability is equal to the probability of $\bigvee_{(i,j)\in E} R(i) \wedge T(j)$, which is F up to variable renaming, proving that $\mathbf{P}(H_0) = \mathbf{P}(F_{H_0,\mathbf{T}}) = \mathbf{P}(F)$. □

This result generalizes to constraints instead of queries. Consider the constraint

$$\Delta = \forall x \forall y (R(x) \wedge S(x,y) \Rightarrow T(y)).$$

From Theorem 3.4, one can easily show that computing the probability $\mathbf{P}(\Delta)$ is #P-hard in the size of the input database, even if the input database is tuple-independent.

The DPLL Family of Algorithms Exact model counting algorithms are based on extensions of the DPLL family of algorithms introduced by Davis and Putnam [1960] and Davis et al. [1962] that were originally designed for satisfiability search. We review them briefly here, and refer to Gomes et al. [2009] for a survey.

A DPLL algorithm chooses a Boolean variable X, uses the Shannon expansion formula $\#F = \#F[X = 0] + \#F[X = 1]$, and computes the number of models of the two residual formulas $\#F[X = 0]$, $\#F[X = 1]$. The DPLL algorithm was initially developed for the Satisfiability Problem. In that case, it suffices to check if one of the two residual formulas $\#F[X = 0]$ or $\#F[X = 1]$ is satisfiable: if the first one is satisfiable, then there is no need to check the second one. When we adapt it to model counting, the DPLL algorithm must perform a full traversal of the search space.

In addition to the basic Shannon expansion step, modern DPLL-based algorithms implement two extensions. The first consists of *caching intermediate results*, to avoid repeated computations of equivalent residual formulas. Before computing $\#F$ the algorithm checks if F is in the cache; if it is in the cache then the algorithm returns $\#F$ immediately; otherwise, it computes $\#F$ using a Shannon expansion, then stores the result in the cache.

The second optimization consists of checking if the clauses of the CNF formula F can be partitioned into two sets F_1, F_2 with no common variables, $F = F_1 \wedge F_2$, in which case they return $\#F = \#F_1 \cdot \#F_2$. The search space for both F_1 and F_2 is significantly smaller, since each has only a subset of the variables. This leads to the following algorithm template, which call a DPLL-based algorithm:

Base case If F is `true`return 1; if F is `false`, return 0.

Cache Read Lookup the pair $(F, \#F)$ in the cache: if found, return $\#F$.

Components If it is possible to write $F = F_1 \wedge F_2$ where F_1, F_2 do not share any common Boolean variables, then compute $\#F = \#F_1 \cdot \#F_2$.

Shannon expansion Otherwise, choose a variable X and compute $\#F = \#F[X = 0] + \#F[X = 1]$.

Cache Write Store the pair $(F, \#F)$ in the cache, and return $\#F$.

We leave it as a simple exercise to adapt this algorithm from model counting to weighted model counting and probability computation.

Concrete implementations differ in their choice of variable order, and the way they trade off the cost and benefit of the cache and components optimizations. We discuss briefly each of these three choices. First, for the variable order, while different heuristics have various tradeoffs, one simple heuristics that always improves the runtime is the *unit clause rule*: if there is a clause consisting of a single literal, i.e. the clause is X (or $\neg X$), then choose X as the next variable to expand, because in that case we do not need to compute $\#F[X = 0]$ (or $\#F[X = 1]$). For example, if $F = (\neg X) \wedge F_1$, then $\#F = \#F[X = 0]$. Second, the design and implementation of the cache is non-trivial. Theoretically, checking if a formula F is in the cache is an co-NP-hard problem, since we need to check, for every formula F' in the cache, whether F is equivalent to F'. In practice, this test is replaced with testing whether the representations of F and F' are syntactically identical. Finally, checking for components also involves a trade-off

between the cost of the test (which amounts to computing connected components) and its potential benefit. To alleviate this cost, and guide the choice of which variable to expand, one can compute a decomposition tree before running the DPLL algorithm [Darwiche, 2000].

3.5 Algorithms for Approximate Model Counting

Let F be a Boolean formula, and ε, δ be two numbers. We say that a randomized algorithm returning a quantity $\tilde{C}$ is an (δ, ε)-approximation of $\#F$ if

$$\mathbf{P}(|\tilde{C} - \#F| > \delta \cdot \#F) < \varepsilon$$

where the probability is taken over the random choices of the algorithm. The meaning of $\tilde{C}$ is that of an estimated value of the exact count $C = \#F$. We say that the algorithm is an *approximation algorithm* for model counting, and define its complexity in terms of the size of the input formula F, and the quantities $1/\varepsilon, 1/\delta$. When the algorithm runs in polynomial time in all three parameters, then call it a Polynomial Time Approximation Scheme (FPTRAS). These definitions for model counting carry over naturally to the probability computation problem, or to the weighted model counting problem, and we omit the straightforward definition.

We will describe below an approximation algorithm based on Monte Carlo simulations. In general, this algorithm is not an FPTRAS. Karp and Luby [1983] have shown that DNF formulas admit an FPTRAS consisting of a modified Monte Carlo simulation. Roth [1996] and Vadhan [2001] proved essentially that no FPTRAS can exists for CNF formulas. More precisely, they proved the following result (building on previous results by Jerrum, Valiant, Vazirani and later by Sinclair): for any fixed $\varepsilon > 0$, given a bipartite graph $E \subseteq [n] \times [n]$, it is NP-hard to approximate $\#F$ within a factor n^{ε}, where $F = \bigwedge_{(i,j)\in E}(X_i \vee X_j)$.

These results directly apply to probabilistic databases.

Theorem 3.5. The following statements hold for tuple-independent probabilistic databases:

- For any Union of Conjunctive Queries Q, the problem *given a tuple-independent probabilistic database* $\mathbf{D}$, *compute* $\mathbf{P}_{\mathbf{D}}(Q)$ admits an FPTRAS.

- Consider the constraint $\Delta = \forall x \forall y (R(x) \wedge S(x,y) \Rightarrow R(y))$. Then, for any $\varepsilon > 0$, the problem *given a tuple-independent probabilistic database* $\mathbf{D}$, *approximate* $\mathbf{P}_{\mathbf{D}}(\Delta)$ *within a relative factor* n^{ε} is NP-hard. Here n is the size of the domain of the probabilistic database. In particular, $\mathbf{P}_{\mathbf{D}}(\Delta)$ does not admit an FPTRAS.

Proof. For any UCQ Q, the lineage $F_{Q,\mathbf{T}}$ is a DNF, hence the first part follows from Karp and Luby [1983]. By setting $\mathbf{P}(S(i,j)) = 0$ when $(i,j) \in E$ and $\mathbf{P}(S(i,j)) = 1$ when $(i,j) \notin E$, the lineage of Δ is, essentially, $\bigwedge_{(i,j)\in E}(R(i) \vee R(j))$, hence the second part follows from Vadhan [2001]. □

The Monte Carlo Algorithm

We will describe the basic Monte Carlo algorithm for approximate probability computation. The input consists of a Boolean formula F, and a probability function $p : \mathbf{X} \to [0,1]$. The algorithm repeatedly chooses a random assignment θ, with probability $p(\theta)$, then returns the fraction of trials where $\theta(F) = \texttt{true}$:

Repeat for $i = 1, N$**:** Compute a random assignment θ by setting randomly and independently for each variable X, $\theta(X) = 0$ with probability $1 - p(X)$, or $\theta(X) = 1$ with probabilities $p(X)$. Denote $Y_i = \theta(F) \in \{0,1\}$.

Return: $\tilde{p} = \sum_i Y_i / N$.

The question that remains is how many steps N do we need to run the algorithm in order to achieve a desired precision, with a desired confidence. To answer this we need the Chernoff/Hoeffding bound:

Theorem 3.6 (Chernoff/Hoeffding [1963])**.** Let $Y_1, Y_2, \ldots, Y_N$ be i.i.d. random variables with values in $\{0,1\}$ and with mean y. Let $\tilde{y} =$

$\sum_i Y_i/N$. Then:

$$\forall \gamma, 1 > \gamma > y: \qquad \mathbf{P}(\tilde{y} \geq \gamma) \leq \exp\left(-N \cdot D(\gamma||y)\right) \tag{3.1}$$

$$\forall \gamma, 0 < \gamma < y: \qquad \mathbf{P}(\tilde{y} \leq \gamma) \leq \exp\left(-N \cdot D(\gamma||y)\right) \tag{3.2}$$

where, forall $0 < \gamma < 1$, $0 < y < 1$, the function

$$D(\gamma||y) \stackrel{\text{def}}{=} \gamma \cdot ln(\frac{\gamma}{y}) + (1-\gamma) \cdot ln(\frac{1-\gamma}{1-y})$$

is the binary relative entropy.

Equation 3.2) follows immediately from Equation 3.1) by replacing Y_i, y, γ with $1-Y_i, 1-y, 1-\gamma$, and the fact that $D(1-\gamma||1-y) = D(\gamma||y)$. The original proof of Equation 3.1) by Hoeffding [1963] uses the moment generating function; an alternative, elementary proof is given by Impagliazzo and Kabanets [2010]. Before applying these inequalities to the Monte Carlo algorithm, we simplify the relative entropy function by using two lower bounds. First, $\forall \delta$, if $0 < (1+\delta)y < 1$ then $D((1+\delta)y||y) \geq y \cdot h(\delta)$, where $h(\delta) \stackrel{\text{def}}{=} (1+\delta)\ln(1+\delta) - \delta$. Second, when δ is small enough, we have[3] $h(\delta) \geq \delta^2/3$ forall $0 \leq \delta \leq 1/2$ and $h(\delta) \geq \delta^2/2$ forall $\delta \leq 0$.

Denoting the true probability of a Boolean formula $p \stackrel{\text{def}}{=} \mathbf{P}(F)$, the estimate $\hat{p}$ after N steps of the Monte Carlo algorithm satisfies:

$$\begin{aligned} 0 < \delta \leq 1/2: \quad \mathbf{P}(\tilde{p} \geq (1+\delta) \cdot p) &\leq \exp\left(-N \cdot D((1+\delta)p||p)\right) \\ &\leq \exp\left(-\frac{Np\delta^2}{3}\right) \\ 0 \leq \delta < 1: \quad \mathbf{P}(\tilde{p} \leq (1-\delta) \cdot p) &\leq \exp\left(-N \cdot D((1-\delta)p||p)\right) \\ &\leq \exp\left(-\frac{Np\delta^2}{2}\right) \end{aligned}$$

[3]The first inequality follows by expanding $D(y(1+\delta)||y) = y(1+\delta)\ln(1+\delta) + [1-y(1-\delta)]\ln[(1-y(1-\delta))/(1-y)]$, then writing the second term as $-[1-y(1-\delta)]\ln[(1-y)/(1-y(1-\delta))] = -[1-y(1-\delta)]\ln[1+(y\delta/(1-y(1-\delta)))] \geq -y\delta$, because $\ln(1+z) \leq z$ forall z. The second inequality follows from Taylor expansion up to the second term: denote $f(\delta) \stackrel{\text{def}}{=} h(\delta) - \delta^2/3$, then $f(0) = f'(0) = 0$, $f''(\delta) = 1/(1+\delta) - 2/3$ which is ≥ 0 for $\delta \leq 1/2$, proving $f(\delta) \geq 0$.

where the probability is over the random choices of the algorithm. Notice that the probability of being off by more than a relative factor δ decreases exponentially in N, but only at a rate permitted by the factor $\delta^2 \cdot p/3$. Thus, we need to run the algorithm for $N = \Theta(\frac{1}{\delta^2 p})$ steps to ensure this probability is low enough. This works well if p is not too small. When we evaluate a Union of Conjunctive Queries (UCQ) over a probabilistic databases, then p is the probability of some output tuple to a query, and is usually reasonably large, because the lineage is a DNF formula where every minterm has a small number of variables; for example, if the query joins three tables, then every minterm in the DNF formula is the conjunction of three Boolean variables, and this implies that $p = \Theta(1)$, i.e. is independent of the size of the probabilistic database. In this case the naive Monte Carlo simulation algorithm requires only $N = \Theta(\frac{1}{\delta^2})$ steps, and works quite well in practice.

Sampling Beyond UCQ For a general DNF formula, p may be exponentially small, for example if $F = X_1 \wedge X_2 \wedge \cdots \wedge X_n$, then $p = 1/2^n$, and one needs to replace the naive Monte Carlo algorithm with the FPTRAS described by Karp and Luby [1983], but UCQ queries do not have such lineage expressions. When approximating the model count of an arbitrary constraint, Theorem 3.5 suggests that no relative approximation guarantees can be obtained. However, given access to an NP-oracle, Ermon et al. [2013] and Chakraborty et al. [2014] compute (δ, ε)-approximations. The desired guarantees can be attained in practice using efficient SAT or optimization solvers. Another approach is to perform Markov Chain Monte Carlo sampling for inference, which can be sped up significantly when the weighted model counting problem is constructed from a relational data model [Jampani et al., 2008, Wick et al., 2010, Van den Broeck and Niepert, 2015].

Dissociation

Gatterbauer and Suciu [2014] describe an approximation method called *dissociation*, which gives guaranteed upper and lower bounds on the probability of a Boolean formula. Fix a formula F with variables $\mathbf{X}$. A dissociation, F', is obtained, intuitively, by choosing a variable X

that occurs only positively in F, then replacing different occurrences of X with fresh variables $X_1, X_2, \ldots$ Formally, denote $\mathbf{Y} = \mathbf{X} - \{X\}$, $\mathbf{X}' = \mathbf{Y} \cup \{X_1, X_2, \ldots\}$, and let $\theta : \mathbf{X}' \rightarrow \mathbf{X}$ be the substitution $\theta(X_i) = X$ forall i, and $\theta(Y) = Y$ for $Y \in \mathbf{Y}$. A Boolean formula F' with the property $F'[\theta] = F$ is called a dissociation of F. Given a probability function $p : \mathbf{X} \rightarrow [0, 1]$, we write $p' : \mathbf{X}' \rightarrow [0, 1]$ for some extension to the variables $X_1, X_2, \ldots$

Theorem 3.7. (1) Suppose the dissociation is conjunctive, meaning that we can write $F' = \bigwedge_i F'_i$ such that X_i occurs only in F'_i (and in no other F'_j, for $j \neq i$). If $p'(X_i) \leq p(X)$, forall i, then $\mathbf{P}(F') \leq \mathbf{P}(F)$; if $\prod_i p'(X_i) \geq p(X)$, then $\mathbf{P}(F') \geq \mathbf{P}(F)$.

(2) Suppose the dissociation is disjunctive, meaning that we can write $F' = \bigvee_i F'_i$ such that X_i occurs only in F'_i. If $p'(X_i) \geq p(X)$ forall i, then $\mathbf{P}(F') \geq \mathbf{P}(F)$; if $\prod_i p'(X_i) \leq p(X)$, then $\mathbf{P}(F') \leq \mathbf{P}(F)$.

For example, we can use the theorem to compute a lower bound on a CNF formula F as follows. If F can be decomposed into $F_1 \wedge F_2$ where F_1, F_2 do not share variables, then compute $\mathbf{P}(F)$ as $\mathbf{P}(F_1)\mathbf{P}(F_2)$. Otherwise, choose a variable X and dissociate it, by giving two fresh names X_1, X_2 in F_1, F_2 respectively; set their probabilities to $p(X_1) = p(X_2) \stackrel{\text{def}}{=} p(X)$. This dissociation is conjunctive, since each clause had at most one copy of X. Repeat this with other variables, until the dissociated formulas F'_1, F'_2 no longer share any common variables, in which case $\mathbf{P}(F') = \mathbf{P}(F'_1)\mathbf{P}(F'_2)$: this probability is a lower bound on $\mathbf{P}(F)$. For a simple illustration, consider $F = (Y \vee X) \wedge (X \vee Z)$. We dissociate it on the variable X obtaining $F' = (Y \vee X_1) \wedge (X_2 \vee Z)$. By setting $p(X_1) = p(X_2) \stackrel{\text{def}}{=} p(X)$ we obtain the lower bound $\mathbf{P}(F) \geq \mathbf{P}(F') = [1 - (1 - p(Y))(1 - p(X))] \cdot [1 - (1 - p(Z))(1 - p(X))]$; by setting $p(X_1) = p(X_2) \stackrel{\text{def}}{=} \sqrt{p(X)}$ we obtain the upper bound $\mathbf{P}(F) \leq \mathbf{P}(F') = [1 - (1 - p(Y))(1 - \sqrt{p(X)})] \cdot [1 - (1 - p(Z))(1 - \sqrt{p(X)})]$. We invite the reader to compute the exact probability $\mathbf{P}(F)$ and check that both inequalities hold.

The notion of dissociation (or relaxation) is also commonly used to simplify probability computation in probabilistic graphical models [Dechter and Rish, 2003, Choi and Darwiche, 2010]. It has deep connections to message-passing inference [Choi and Darwiche, 2006].

Approximate DPLL

Olteanu et al. [2010] describe a simple heuristics that terminates the DPLL early, and returns lower and upper bounds on the probability rather than the exact probability. When it terminates, it approximates the probability of the unprocessed residual formula by using these bounds:

$$\max_{i=1} \mathbf{P}(F_i) \leq \mathbf{P}(\bigvee_{i=1} F_i) \leq \sum_{i=1} \mathbf{P}(F_i)$$
$$\sum_{i=1} \mathbf{P}(F_i) - (n-1) \leq \mathbf{P}(\bigwedge_{i=1} F_i) \leq \min_{i=1} \mathbf{P}(F_i)$$

During a Shannon expansion step, on a variables X, $\mathbf{P}(F) = (1-p)\mathbf{P}(F_0) + p\mathbf{P}(F_1)$, it propagates the lower/upper bounds of F_0, F_1 to lower/upper bounds of F by setting the lower bound $L = (1-p)L_0 + pL_1$ and the upper bound $U = (1-p)U_0 + pU_1$. Thus, at each step of the DPLL search procedure, this modified algorithm can provide a lower and upper bound on the true probability of F, and the search can be stopped as soon as these two bounds are within a desired precision.

A related technique is used in probabilistic logic programming: unfold the query to a finite depth, and assume that the remaining recursions either all succeed, or all fail, to obtain an upper and lower bound [Poole, 1993, De Raedt et al., 2007, Vlasselaer et al., 2015].

4

Lifted Query Processing

Lifted query processing on probabilistic databases, in short lifted inference, is a generic term given to a variety of techniques to compute the query probability without first grounding the query on the database. These techniques have been developed in parallel in the AI literature, and in the database literature. The notion of computing the probability directly from the first-order sentence was introduced by Poole [2003], and the actual term *lifted inference* was coined by de Salvo Braz et al. [2005]. In the database literature, the idea of *extensional query evaluation* can be traced to early work by Barbará et al. [1992] and Fuhr [1993]; the term extensional semantics seems to occur first in Fuhr and Rölleke [1997], and is derived from the extensional semantics discussed by Pearl [1988].

All lifted inference techniques described in the literature run in polynomial time in the size of the probabilistic database and therefore they cannot work in general, since the query evaluation problem is #P-hard in general. Instead, lifted inference works only for certain classes of queries and/or databases. There is no formal definition of the term lifted inference. Van den Broeck [2011] proposes a definition that captures one common feature of all exact lifted inference meth-

ods: it defines lifted inference to be any method that runs in polynomial time in the size of the domain (or database). However, there is a second feature shared by all lifted inference methods, which has eluded a formal definition, namely they all use the structure of the first-order formula to guide the lifted inference. In other words, lifted inference techniques start by examining the first-order formula, and making a plan on how to proceed in the evaluation, before the data is touched, if at all. For that reason, lifted inference is quite similar to traditional query evaluation techniques on deterministic databases, which also construct a relational plan first, then evaluate the query by following the plan. In fact, most lifted inference techniques can be described in a similar way: first construct a query plan, then evaluate the plan on the probabilistic database; moreover the evaluation can often be pushed inside the database engine.

The lifted inference techniques exploit some simple properties in probability theory: independence of events, the inclusion/exclusion formula, and de Finetti's theorem for symmetric probability spaces. Compare this with the DPLL family of model counting algorithms for Boolean formulas, which exploit only the Shannon expansion and the independence formulas; we will show in the next chapter that lifted inference algorithms are provably more powerful than grounded inference based on the DPLL family of algorithms.

In this chapter we start by describing lifted inference as query plans, then discuss a complete set of lifted rules for Unions of Conjunctive Queries, or, more generally, unate first-order logic with a single type of quantifier ($\forall$ or $\exists$). Readers interested in more details about this part (Sections 4.1-4.4) are referred to the book by Suciu et al. [2011], and to Dalvi and Suciu [2012] for the proof of the dichotomy theorem. Then, we discuss extensions to queries with negations, and extensions that work on a larger class of queries but restrict the database.

4.1 Extensional Operators and Safe Plans

In this section we consider *set-valued* queries (§2.3), whose answer is a relation (set of tuples); Boolean queries are a special case. By Definition 2.2, the semantics of a set-valued query over a probabilistic database is a set of tuples annotated with probabilities. We will assume that all input relations are tuple-independent, or BID tables.

A query plan is an algebraic expression where each operator is one of: join, selection, projection, union, difference, and aggregate. Almost all database systems today process a query by first converting it into a query plan, then evaluating the plan over the database.

Extensional Operators

A query plan can be extended to compute probabilities. Assume that each intermediate relation in the query plan has an attribute p representing the probability that the tuple appears in that intermediate relation. Then, each relational operator is extended as follows; these operators are called *extensional operators*:

- For a join operator $R \bowtie S$, compute p assuming that every pair of tuples that join are independent. In other words, for every pair of joining tuples, its probability attribute p is computed as $R.p * S.p$.
- For a selection operator, $\sigma(R)$, simply propagate the probability attribute for all tuples that satisfy the selection criterion.
- For a projection $\Pi_A(R)$, compute p assuming that all tuples that are combined into a single tuple during duplicate elimination are independent events. In other words, the probability attribute of an output tuple is $1 - (1 - t_1.p)(1 - t_2.p) \cdots$ where $t_1, t_2, \ldots \in R$ are all tuples that have a common value of the attribute(s) A. A variant of the projection operator is to assume that these tuples are disjoint probabilistic events, and in that case return $t_1.p + t_2.p + \cdots$ We call the former an *independent* project and the latter a *disjoint* project.

- For a union $R \cup S$, compute p assuming that the tuples in R and S are independent. In other words, the output probability of a tuple is either $R.p$, if it occurs only in R, or $S.p$ if it occurs only in S, or $1 - (1 - R.p)(1 - S.p)$ if it occurs in both.
- For a difference operator $R - S$, compute p assuming their tuples are independent events. In other words the output probability is either $R.p$ if the tuple occurs only in R, or $R.p * (1 - S.p)$ if it occurs in both.

An *extensional query plan* is a standard query plan with operators $\bowtie, \sigma, \Pi, \cup, -$, where each operator is interpreted as an extensional operator. Each projection operator Π needs to be annotated to indicate whether it is an independent project, or a disjoint project; by default we assume it is an independent project. When evaluated over a probabilistic database $\mathbf{D}$, the plan returns a probability for each query answer: this is called *extensional query evaluation*. These output probabilities may or may not be the correct output probabilities according to Definition 2.2; when they are correct for any probabilistic database, then we call the plan a *safe plan*. We discuss safe plans next.

Safe Plans

Fix a relational schema for a probabilistic database, which specifies for each table name whether it is a tuple-independent table or a BID table; in the latter case the schema also specifies the key(s) defining the groups of disjoint tuples (cf. §2.2.1).

Definition 4.1. Let Q be a set-valued query. A *safe plan* for a query Q is an extensional query plan that computes correct output probabilities (according to Definition 2.2), on any input probabilistic database.

We illustrate with an example.

Example 4.1. Assume both relations R, S are tuple-independent, and consider the query:

$$\texttt{Q(z):-R(z,x)} \wedge \texttt{S(x,y)}$$

The equivalent SQL query is

```
select distinct R.z from R, S where R.x = S.x
```

Figure 4.1a shows a small tuple-independent probabilistic database. The only possible answer to this query is $\{c\}$, since the attribute z has a single value c in the relation R. Thus, in any possible world the answer to the query is either $\{c\}$ or $\emptyset$. Our goal is to compute the probability that c is in the answer, and, for that, at least one of the following two events must hold:

- The tuple (c, a_1) occurs in R, and at least one of the tuples $(a_1, b_1), (a_1, b_2)$ occurs in S: the probability of this event is $p_1(1 - (1 - q_1)(1 - q_2))$.
- Tue tuple (c, a_2) occurs in R, and at least one of the tuples $(a_2, b_3), (a_2, b_4), (a_2, b_5)$ occurs in S: the probability of this event is $p_2(1 - (1 - q_3)(1 - q_4)(1 - q_5))$.

Since these two events are independent (they refer to disjoint sets of independent tuples), the probability that c is in the answer is:

$$1 - [1 - p_1(1 - (1 - q_1)(1 - q_2))] \cdot [1 - p_2(1 - (1 - q_3)(1 - q_4)(1 - q_5))]$$

To compute the query on a traditional database, any modern query engine would produce a query plan like that in Figure 4.1b: it first joins R and S on the attribute x, then projects the result on the attribute z. We assume that the project operator includes duplicate elimination. If we extend each operator to manipulate explicitly the probability attribute p, then we obtain the intermediate result and final result shown in the figure (the actual tuples are dropped and only the probabilities are shown, to reduce clutter). As we can see the final result is wrong. The reason is that duplicate elimination incorrectly assumed that all tuples that have $z = c$ are independent: in fact the first two such tuples depend on (c, a_1) and the next three depend on (c, a_2) (see the repeated probabilities p_1, p_1 and p_2, p_2, p_2 in the figure). Thus, the plan in Figure 4.1b is not safe.

In contrast, the plan in Figure 4.1c is safe, because it computes the output probabilities correctly; the figure shows the computation only

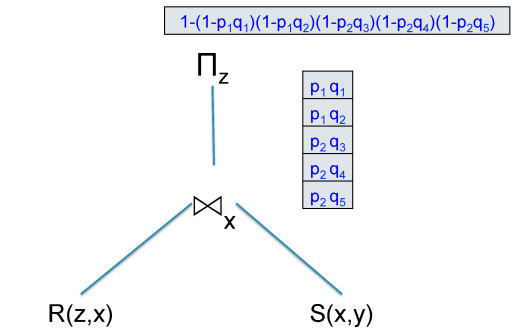
R :

z	x	p
c	a_1	p_1
c	a_2	p_2
c	a_3	p_3

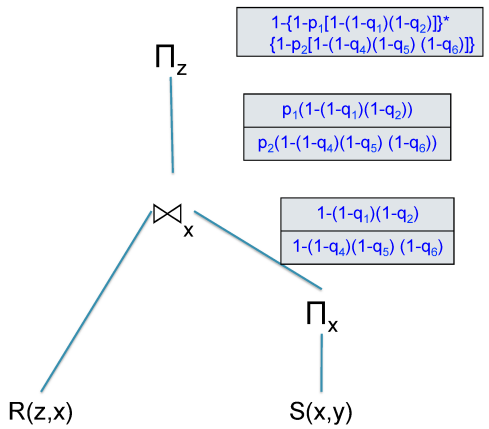
S :

x	y	p
a_1	b_1	q_1
a_1	b_2	q_2
a_2	b_3	q_3
a_2	b_4	q_4
a_2	b_5	q_5

(a) Probabilistic database



(b) Unsafe plan



(c) Safe plan

Figure 4.1: Example query plans. Intermediate partial results show the probabilities.

on our toy database, but the plan is correct on *any* input database. It starts by projecting out the redundant attribute y in $S(x, y)$ and doing duplicate elimination, before joining the result with R, then continues like the previous plan. The two plans (b) and (c) are equivalent over deterministic databases, but when the join and projection operators are extended to manipulate probabilities, then they are no longer equivalent. Notice that a SQL engine would normally not choose (c) over (b), because the extra cost of the duplicate elimination is not justified. Over probabilistic databases, however, the two plans are different, and the latter returns the correct probability, as shown in the figure. We invite the reader to verify that this plan returns the correct output probabilities for *any* tuple independent probabilistic relations R and S.

A safe query plan can be executed directly in the database engine, which is often much more efficient than a custom implementation. For example, the MayBMS system by Antova et al. [2007], Olteanu et al. [2009] modified the postgres source code converting each operator into an extensional operator. A more portable, slightly less efficient alternative, is to encode the safe plan back into SQL, with aggregate operators for manipulating the probability attributes, and have them executed on any standard database engine. Safe plans, when they exists, are almost as efficient as traditional query plans on deterministic databases.

Not every query has a safe plan. For example, $H_0 = \exists x \exists y R(x) \wedge S(x, y) \wedge T(y)$ has no safe plan: neither $P_1 \stackrel{\text{def}}{=} \Pi_\emptyset(\Pi_y(R(x) \bowtie_x S(x, y)) \bowtie_y T(y))$ nor $P_2 \stackrel{\text{def}}{=} \Pi_\emptyset(R(x) \bowtie_x \Pi_x(S(x, y) \bowtie_y T(y)))$ is safe. This is to be expected, since we have shown in Theorem 3.4 that the complexity of computing $P(H_0)$ is #P-hard.

Unsafe Plans

If we evaluate an extensional, unsafe plan, are the resulting probabilities of any use? Somewhat surprisingly, Gatterbauer and Suciu [2015] show that *every* extensional plan for a conjunctive query without self-joins returns an upper bound on the true probabilities. In other words,

even if the plan is unsafe, it always returns an upper bound on the true probability. This follows from Theorem 3.7 (2) by observing that every extensional plan computes the exact probability of some dissociation of the lineage. Indeed, any projection operator treats repeated copies of the same tuple as distinct random variables, which means that it dissociates the random variable associated to the tuple. For example, both plans P_1 and P_2 above return upper bounds on the probability of H_0, and obviously can be computed in polynomial time in the size of the input database. In general, by considering *all* plans for the query, and taking the minimum of their probabilities one obtains an even tighter upper bound on the true probabilities than a single plan. Some plans are redundant and can be eliminated from the enumeration of all plans, because they are dominated by tighter plans. For example, H_0 admits a third plan, $P_3 \stackrel{\text{def}}{=} \Pi_\emptyset(R(x) \bowtie S(x,y) \bowtie T(y))$, but Theorem 3.7 (2) implies that $\text{eval}(P_1) \leq \text{eval}(P_3)$ and $\text{eval}(P_2) \leq \text{eval}(P_3)$, because the lineage of P_3 is a dissociation of the lineage of both P_1 and P_2; hence P_3 is dominated by P_1 and P_2 and does not need to be considered when computing the minimum of all probabilities. The technique described in Gatterbauer and Suciu [2015] is quite effective for conjunctive queries without self-joins, but it is open whether it can be extended to more general queries.

To summarize, some queries admit safe plans and can be computed as efficiently as queries over standard databases; others do not admit safe plans. The practical question is: which queries admit safe plans? And if a query admits a safe plan, how can we find it? We address this issue in the rest of this chapter.

4.2 Lifted Inference Rules

We describe here an alternative formalism for lifted query evaluation, by using rules. This formalism is equivalent to safe query plans, but simpler to describe rigorously and prove properties about. In this section and in the remainder of this chapter, unless otherwise stated we will assume that the input probabilistic database is tuple-independent, and that all queries are Boolean queries and are expressed as a first-order sentence.

Call two Boolean formulas F_1, F_2 *independent* if, for any probability function $p : \mathbf{X} \to [0,1]$, $p(F_1 \wedge F_2) = p(F_1)p(F_2)$. We have seen in §3.1 that F_1, F_2 are independent iff they depend on disjoint sets of variables. Similarly, given two FO sentences Q_1, Q_2 we say that Q_1, Q_2 are independent if, for any finite domain D, the lineage expressions $F_{Q_1,D}$ and $F_{Q_2,D}$ are independent. If Q is a formula with a single free variable x, then we call x a *separator variable* if for any two distinct constants a, b the two queries $Q[a/x]$ and $Q[b/x]$ are independent. If $Q = \exists x Q_1$ or $Q = \forall x Q_1$, then, with some abuse, we say that x is a separator variable in Q to mean that it is a separator variable in Q_1.

We can now describe the lifted inference rules. Let $\mathbf{D} = (\mathbf{T}, p)$ a tuple-independent probabilistic database, where $\mathbf{T}$ is an ordinary database and $p : \mathbf{T} \to [0,1]$. Denote D the domain of the database. We extend the probability function to $p : \text{Tup}(D) \to [0,1]$ by setting $p(t) = 0$ when $t \notin \mathbf{T}$. Then for every first-order sentence Q:

Ground Tuple If $Q = t$ (a ground tuple) then $\mathbf{P}(Q) = p(t)$.

Negation If $Q = \neg Q_1$ then $\mathbf{P}(Q) = 1 - \mathbf{P}(Q_1)$.

Join If $Q = Q_1 \wedge Q_2$ and Q_1, Q_2 are independent, then $\mathbf{P}(Q) = \mathbf{P}(Q_1)\mathbf{P}(Q_2)$.

Union If $Q = Q_1 \vee Q_2$ and Q_1, Q_2 are independent, then $\mathbf{P}(Q) = 1 - (1 - \mathbf{P}(Q_1)) \cdot (1 - \mathbf{P}(Q_2))$.

Universal Quantifier If $Q = \forall x Q_1$ and x is a separator variable in Q_1 then $\mathbf{P}(Q) = \prod_{a \in D} Q_1[a/x]$.

Existential Quantifier If $Q = \exists x Q_1$ and x is a separator variable in Q_1 then $\mathbf{P}(Q) = 1 - \prod_{a \in D}(1 - Q_1[a/x])$.

Inclusion/Exclusion 1 If $Q = Q_1 \vee Q_2$ then $\mathbf{P}(Q) = \mathbf{P}(Q_1) + \mathbf{P}(Q_2) - \mathbf{P}(Q_1 \wedge Q_2)$.

Inclusion/Exclusion 2 If $Q = Q_1 \wedge Q_2$ then $\mathbf{P}(Q) = \mathbf{P}(Q_1) + \mathbf{P}(Q_2) - \mathbf{P}(Q_1 \vee Q_2)$.

All rules are simple applications of independence, except for the last two, which are simple applications of the inclusion/exclusion principle. We invite the reader to check that the rules are correct (sound) w.r.t. the standard query probability (Definition 2.2). For example, the universal quantifier rule is correct, because the lineage of $Q = \forall x Q_1$ is the Boolean formula $F_{Q,D} = \bigwedge_{a \in D} F_{Q_1[a/x],D}$. Since x is a separator variable, for any two constants $a, b \in D$ the two formulas $F_{Q_1[a/x],D}$ and $F_{Q_1[b/x],D}$ are independent, hence they have disjoint sets of variables. This implies that $\{F_{Q_1[a/x],D} \mid a \in D\}$ is a set of independent events, since any two have disjoint sets of Boolean variables, and this implies $\mathbf{P}(F_{Q,D}) = \prod_{a \in D} \mathbf{P}(F_{Q_1[a/x],D})$.

To convert these rules into an algorithm, we need to discuss how to check independence of two sentences Q_1, Q_2 and how to check whether x is a separator variable. These problems are hard in general, as we explain in §4.4. In the lifted inference algorithm, we use instead the following syntactic conditions, which are sufficient but, in general, not necessary.

Definition 4.2. (1) Two sentences Q_1, Q_2 are syntactically independent if they use disjoint sets of relational symbols. (2) Let Q be a formula with one free variable x. We say that x is a syntactic separator variable if for every relational symbol R there exists a number $i_R \in [arity(R)]$ such that every atom in Q that refers to R contains x on position i_R; in particular every atom must contain x.

The following is easily verified.

Lemma 4.1. If Q_1, Q_2 are syntactically independent then they are independent. If x is a syntactic separator variable for Q, then it is a separator variable.

For example $Q_1 = \exists x R(x)$ is syntactically independent of $Q_2 = \exists y \exists z S(y, z)$, and x is a syntactic separator variable for the formula $R(x) \wedge (\exists y S(x, y)) \vee T(x) \wedge (\exists z S(x, z))$ (every atom contains x and both S-atoms contain x on the same position 1).

The lifted inference rules represent a non-deterministic algorithm for computing $\mathbf{P}(Q)$, for any FO sentence Q. Start from Q, and repeatedly apply the lifted inference rules, reducing $\mathbf{P}(Q)$ to simpler queries

$\mathbf{P}(Q')$, until we reach ground atoms, and then return the probability of the ground atom, which is obtained directly from the database. If we reach a sentence Q' where no rule applies, then we are stuck, and lifted inference failed on our query.

We illustrate with a simple example, computing the probability of the constraint $\Gamma = \forall x \forall y(S(x,y) \Rightarrow R(x))$:

$$\begin{aligned}
\mathbf{P}(\forall x \forall y(S(x,y) \Rightarrow R(x))) &= \prod_{a \in D} \mathbf{P}(\forall y(S(a,y) \Rightarrow R(a))) \\
&= \prod_{a \in D} \mathbf{P}(\forall y(\neg S(a,y) \vee R(a))) \\
&= \prod_{a \in D} \mathbf{P}((\forall y \neg S(a,y)) \vee R(a)) \\
&= \prod_{a \in D} 1 - (1 - \mathbf{P}((\forall y \neg S(a,y)))(1 - p(R(a)))) \\
&= \prod_{a \in D} 1 - (1 - \prod_{b \in D} (1 - p(S(a,b)))(1 - p(R(a))))
\end{aligned}$$

In the last expression all probabilities are for ground atoms, and these can be obtained using the function p, or, in practice, looked up in an attribute of the relation. We invite the reader to check that in both steps where we applied the universal quantifier rule, the variable that we eliminated was a syntactic separator variable.

When the rules succeed, then we compute $\mathbf{P}(Q)$ in polynomial time in the size of the input domain: more precisely in time $O(n^k)$ where n is the size of the domain and k is total number of variables. Therefore, the lifted inference rules will not work on queries whose complexity is #P-hard. For example, the query $H_0 = \exists x \exists y R(x) \wedge S(x,y) \wedge T(y)$ is #P-hard (by Theorem 3.4). It is easy to check that no lifted inference rule applies here: the query is not the conjunction of two sentences, so we cannot apply the join-rule, and neither x nor y is a separator variable (because none of them occurs in all atoms). In general, it will be the case that for some first-order sentences the lifted inference rules will not be able to compute the query.

An important question is to characterize the class of queries whose probability can be computed using only the lifted inference rules. We say that such a query is *liftable*. A query can be computed using lifted

inference rules iff it admits a safe query plan, and therefore we also call such a query a *safe* query. When Q is liftable, then $\mathbf{P}(Q)$ can be computed in polynomial time in the size of the database. If the converse also holds, then the rules are complete. We describe below some fragments of first-order sentences for which the rules are complete: more precisely, if the rules fail to compute the probability of some query Q, then $\mathbf{P}(Q)$ is provably #P-hard in the size of the input database.

4.3 Hierarchical Queries

Definition 4.3. Let Q be first-order formula. For each variable x denote $at(x)$ the set of atoms that contain the variable x. We say that Q is *hierarchical* if forall x, y one of the following holds: $at(x) \subseteq at(y)$ or $at(x) \supseteq at(y)$ or $at(x) \cap at(y) = \emptyset$.

The definition is syntactic. It is quite possible to have two equivalent FO sentences, one hierarchical the other non-hierarchical: for example, $\exists x \exists y \exists z R(x, y) \wedge S(x, z)$ is hierarchical and is equivalent to $\exists x \exists y \exists z \exists u R(x, y) \wedge S(x, z) \wedge S(u, z)$ which is non-hierarchical, because the two sets $at(x) = \{R(x, y), S(x, z)\}$ and $at(z) = \{S(x, z), S(u, z)\}$ overlap without one being contained in the other. Some queries are not equivalent to any hierarchical sentence: for example the query $H_0 = \exists x \exists y R(x) \wedge S(x, y) \wedge T(y)$ is non-hierarchical, because $at(x) = \{R(x), S(x, y)\}$, $at(y) = \{(S(x, y), T(y)\}$, and the reader may check that no equivalent conjunctive query is hierarchical.

Recall the definition of a *Boolean conjunctive query*: it is a sentence of the form

$$Q = \exists x_1 \exists x_2 \cdots \exists x_k R_1(t_1) \wedge R_2(t_2) \wedge \cdots \wedge R_\ell(t_\ell)$$

where each $R_j(t_j)$ is a relational atoms. Conjunctive queries correspond to $FO^{\exists,\wedge}$. We say that Q is *without self-joins* if all relational symbols $R_1, \ldots, R_\ell$ are distinct. Other terms used in the literature for conjunctive queries without self-joins are *simple conjunctive queries*, or a *non-repeating conjunctive queries*.

The dual of a conjunctive query without self-joins is a positive clause without repeated symbols:

$$\Gamma = \forall x_1 \forall x_2 \cdots \forall x_k (R_1(t_1) \vee R_2(t_2) \vee \cdots \vee R_\ell(t_\ell))$$

where the relational symbols $R_1, \ldots, R_\ell$ are distinct.

Theorem 4.2 (Small Dichotomy Theorem)**.** Dalvi and Suciu [2007b] Let Q be a conjunctive query without self-joins. Then:

- If Q is hierarchical, then $\mathbf{P}(Q)$ is in polynomial time, and can be computed using only the lifted inference rules for join and existential quantifier.
- If Q is not hierarchical, then $\mathbf{P}(Q)$ is #P-hard in the size of the database.

A similar statement holds for positive clauses without repeated relational symbols.

Proof. Suppose Q is hierarchical. Let x be any variable for which $at(x)$ is maximal. Case 1: $at(x)$ consists of all atoms in Q. Then we can write $Q = \exists x Q_1$ where x is a separator variable, and we can apply the $\exists$-lifted inference rule $\mathbf{P}(Q) = 1 - (1 - \prod_{a \in D} Q_1[a/x])$; the claim follows by induction, since $Q[a/x]$ is also hierarchical, for any constant a. Case 2: there exists some atom not in $at(x)$. Then the atoms in $at(x)$ and those not in $at(x)$ do not share any logical variable: if they shared a variable y then $at(x) \cap at(y) \neq \emptyset$, and therefore $at(y) \subseteq at(x)$ (by maximality of x), which contradicts the fact y occurs in some atom that is not in $at(x)$. Then we can write the query as $Q = Q_1 \wedge Q_2$ where both Q_1 and Q_2 are existentially quantified sentences. Since the query has no self-joins, we can apply the join rules and write $\mathbf{P}(Q) = \mathbf{P}(Q_1) \cdot \mathbf{P}(Q_2)$; the claim follows by induction since both Q_1, Q_2 are hierarchical.

Suppose Q is not hierarchical. We prove that it is #P hard by reduction from $H_0 = \exists x \exists y R(x) \wedge S(x, y) \wedge T(y)$, which is #P-hard by Theorem 3.4. Consider a probabilistic database instance over three relations $R(x), S(x, y), T(y)$, where we want to compute $\mathbf{P}(H_0)$. Consider any non-hierarchical query Q. By definition, Q contains two variables x', y'

s.t. $at(x')$, $at(y')$ are overlapping but none contains the other. Thus, Q has three atoms $R' \in at(x') - at(y')$, $S' \in at(x') \cap at(y')$, $T' \in at(y') - at(x')$. The three atoms are $R'(x', \ldots) \wedge S'(x', y', \ldots) \wedge T'(y', \ldots)$, i.e. R' contains the variable x' and possibly others, but does not contain the variable y', similarly for T', while S' contains both x', y'. Given the input probabilistic database $R(x), S(x, y), T(y)$, we construct an instance for $R'(x', \ldots)$ by extending $R(x)$ with the extra attributes and filling them with some fixed constant; similarly, construct S', T' from S, T, where the extended attributes are filled with the same constant. The new relations R', S', T' are therefore in 1-1 correspondence with R, S, T: they have the same cardinalities, and corresponding tuples have the same probabilities. For all other relations in Q, define their instance to be the cartesian product of the domain, and set their probabilities to be 1. It is easy to check that $\mathbf{P}(Q) = \mathbf{P}(H_0)$, proving that the query evaluation problem for H_0 can be reduced to that for Q; hence Q is #P-hard. □

The theorem establishes a dichotomy for the class of conjunctive queries without self-joins: every query is either liftable (and thus in polynomial time) or provably #P-hard, and the separation is given precisely by hierarchical queries. The same holds for positive clauses without repeated relational symbols. For example, $\forall x \forall y \forall z (R(x, y) \vee S(x, z) \vee T(x))$ is hierarchical, hence in PTIME, while $\forall x \forall y \forall z (R(x, y) \vee S(y, z) \vee T(z))$ is non-hierarchical, hence #P-hard. In fact, the dichotomy result also holds immediately for any clause without repeated relational symbols (not necessarily positive); we chose to state Theorem 4.2 only for positive sentences for historical reasons. Next, we will describe a non-trivial extension of the dichotomy theorem, to a larger class of first-order sentences.

4.4 The Dichotomy Theorem

In this section we state a more general dichotomy theorem, which applies to a larger fragment of first-order logic. Without loss of generality, in this section we assume that the formulas use only the connectives $\neg, \exists, \forall, \vee, \wedge$ (for example $Q_1 \Rightarrow Q_2$ can be rewritten as $\neg Q_1 \vee Q_2$)

and that all occurrences of the $\neg$ operator are pushed down to the atoms, using De Morgan's laws. We will consider various fragments of FO. If $S \subseteq \{\neg, \exists, \forall, \vee, \wedge\}$, then FO^S denotes the fragment of FO restricted to the operations in S. For example, $FO^{\exists,\wedge}$ is equivalent to Conjunctive Queries (CQ), $FO^{\exists,\vee,\wedge}$ is the positive, existential fragment of FO, and equivalent to Unions of Conjunctive Queries (UCQ), while $FO^{\forall,\vee,\wedge}$ is positive universal fragment of FO, and $FO^{\neg,\exists,\forall,\vee,\wedge}$ is the full first-order logic, which we denote FO.

Call a first-order formula *unate* if every relational symbol occurs either only positively, or only negated. For example, the first sentence below is unate, the second is not:

$$\Gamma_1 = \forall x \forall y (\neg R(x) \vee \neg S(x,y)) \wedge (\neg S(x,y) \vee T(y)) \qquad \text{// unate}$$
$$\Gamma_2 = \forall x \forall y (\neg R(x) \vee S(x,y)) \wedge (\neg S(x,y) \vee T(y)) \qquad \text{// non unate}$$

We denote $FO^{\neg un,\exists,\forall,\vee,\wedge}$ the unate fragment of first-order logic; similarly $FO^{\neg un,\exists,\vee,\wedge}$ and $FO^{\neg un,\exists,\forall,\vee,\wedge}$ are the existential and universal fragment of unate first-order logic respectively.

The dichotomy theorem that we state below says that every query in $FO^{\neg un,\exists,\vee,\wedge}$ is in polynomial time when the lifted inference rules in §4.2 succeed, or is provably #P-hard when those rules fail, and similarly for $FO^{\neg un,\forall,\vee,\wedge}$. The practical significance of this result is that the inference rules are complete: there is no need to search for more rules, since all queries than can be possibly computed in PTIME are already handled by the existing rules.

The Independence Test

However, we need some preparation in order for this result to hold. Consider the join-rule: if $Q = Q_1 \wedge Q_2$ and Q_1, Q_2 are independent, then $\mathbf{P}(Q) = \mathbf{P}(Q_1)\mathbf{P}(Q_2)$. In the algorithm we replaced the test "Q_1, Q_2 are independent" with "Q_1, Q_2 are syntactically independent", which is sufficient, but not necessary. Obviously, if Q_1, Q_2 are independent but not syntactically independent then the inference rule fails, and the dichotomy theorem cannot hold as stated: the query may be in PTIME (assuming both $\mathbf{P}(Q_1)$ and $\mathbf{P}(Q_2)$ are in PTIME) but we cannot apply the lifted inference rule because we are unable to check

independence. To prove a dichotomy result, we need a necessary and sufficient test for independence, and the same for a separator variable.

Checking independence is hard in general. If Q_1, Q_2 are two sentences in FO, then checking if they are independent is undecidable in general, by Trakhtenbrot's undecidability theorem for finite satisfiability[1]. Miklau and Suciu [2007] showed that, if Q_1, Q_2 are restricted to Conjunctive Queries, then checking independence is Π_2^p-complete. These observations carry over to the test of whether x is a separator variable.

It turns out that one can rewrite the query into a form in which syntactic independence is both necessary and sufficient for independence, and similarly for a separator variable.

Definition 4.4. (1) An FO sentence Q is *shattered* if it does not contain any constants. (2) An FO sentence Q is *ranked* if there exists a total order σ on its variables such that whenever x_i, x_j occur in the same atom and x_i occurs before x_j, then x_i strictly precedes x_j in the order σ; in particular, no atom contains the same variable twice.

We briefly review the notion of a *minimal query*, c.f. Abiteboul et al. [1995]. A conjunctive query Q is called *minimal* if no equivalent query with fewer atoms exists; for example $\exists x \exists y R(x) \wedge R(y)$ is not minimal, since it is equivalent to $\exists x R(x)$. Every conjunctive query Q is equivalent to a minimal query, which is unique up to isomorphism, and is also called the *core* of Q. A Union of Conjunctive Query $\bigvee_i Q_i$ is minimal if each Q_i is minimal, and whenever a logical implication $Q_i \Rightarrow Q_j$ holds, then $i = j$. Intuitively, if $Q_i \Rightarrow Q_j$ holds then Q_i is redundant in the UCQ and can be dropped. Every UCQ query is equivalent to a minimal UCQ query, which is unique up to isomorphism. We call a sentence in $FO^{\neg un,\exists,\vee,\wedge}$ minimal if it is minimal when viewed as a UCQ, by replacing each negated atom with a positive atom. Similarly, we call a sentence in $FO^{\neg un,\forall,\vee,\wedge}$ minimal if its dual (obtained by replacing $\forall, \wedge, \vee$ with $\exists, \vee, \wedge$ respectively) is minimal.

[1] Trakthenbrot's theorem says that the problem *given a FO sentence ϕ, does ϕ have a finite model?* is undecidable. Consider two sentences $Q_1 = \exists x R(x)$ and $Q_2 = \phi \wedge \exists x R(x)$: they are independent iff ϕ is not satisfiable, and hence checking independence is undecidable.

We invite the reader to check the following to lemmas:

Lemma 4.3. (1) If Q_1, Q_2 are two shattered, ranked and minimal sentences in $FO^{\neg un,\exists,\vee,\wedge}$ (or $FO^{\neg un,\forall,\vee,\wedge}$), then syntactic independence is equivalent to independence. (2) If Q is in $FO^{\neg un,\exists,\vee,\wedge}$ (or $FO^{\neg un,\forall,\vee,\wedge}$) is shattered, ranked, and minimal, and has a single free variable x, then x is a syntactic separator variable iff it is a separator variable.

Lemma 4.4. For any sentence Q in $FO^{\neg un,\exists,\vee,\wedge}$ (or $FO^{\neg un,\forall,\vee,\wedge}$) the query evaluation problem $\mathbf{P}(Q)$ can be reduced in polynomial time to the query evaluation problem $\mathbf{P}(Q')$ of a shattered and ranked query Q'.

Given these facts, the lifted inference rules need to be amended as follows. To compute a query Q we start shattering and ranking Q. This only needs to be done once, before we apply the lifted inference rules. Next, we apply the lifted inference rules: when we check whether Q_1, Q_2 are independent (or whether x is a separator variable in Q), then we minimize the queries first, then apply the syntactic independence (or syntactic separation) test.

We illustrate the shattering and ranking in Lemma 4.4 on two examples. The reader is invited to generalize from here (and prove the two lemmas above), or to check further details in Suciu et al. [2011].

Example 4.2. Let

$$Q_1 = \exists x \exists y (R(a, y) \wedge R(x, b))$$

where a, b are two constants in the domain. There is no separator variable since none of the two variables occurs in both atoms. Writing Q_1 as a conjunction of $\exists y R(a, y)$ and $\exists x R(x, b))$ doesn't help either, since these queries are dependent (their lineages both depend on the tuple $R(a, b)$). Instead, we rewrite the query into a shattered query Q_1' as

follows. We split the relation R into four relations:

$$R_{**} \stackrel{\text{def}}{=} \{(x,y) \mid (x,y) \in R, x \neq a, y \neq b\}$$
$$R_{a*} \stackrel{\text{def}}{=} \{y \mid (a,y) \in R, y \neq b\}$$
$$R_{*b} \stackrel{\text{def}}{=} \{x \mid (x,b) \in R, x \neq a\}$$
$$R_{ab} \stackrel{\text{def}}{=} \{() \mid (a,b) \in R\}$$

and rewrite the query as:

$$Q_1' = (\exists x R_{*b}(x) \wedge \exists y R_{a*}(y)) \vee R_{ab}()$$

The new query is shattered, i.e. does not contain any constants. It is easily verified that Q_1 and Q_1' are equivalent, hence $\mathbf{P}(Q_1) = \mathbf{P}(Q_1')$. To compute $\mathbf{P}(Q_1')$ we notice that $\exists x R_{*b}(x) \wedge \exists y R_{a*}(y)$ is syntactically independent of $R_{ab}()$, and furthermore $\exists x R_{*b}(x)$ and $\exists y R_{a*}(y)$ are syntactically independent, hence:

$$\mathbf{P}(Q_1') = 1 - [1 - \mathbf{P}(\exists x R_{*b}(x))\mathbf{P}(\exists y R_{a*}(y))] \cdot [1 - p(R(a,b))]$$

Example 4.3. Consider now the following query

$$Q_2 = \exists x \exists y (S(x,y) \wedge S(y,x))$$

This query is not ranked, because x, y occur in opposite orders in $S(x,y)$ and in $S(y,x)$. No lifted inference rule applies; for example x is not a separator variable since it occurs on different positions in the two atoms (and one may verify that the queries $\exists y(S(a,y) \wedge S(y,a))$ and $\exists y(S(b,y) \wedge S(y,b))$ are dependent, since their lineage expressions both depend on the ground atoms $S(a,b)$ and $S(b,a)$). Instead, we rewrite the query as follows. First, we partition the relation S into three new relations:

$$S_< \stackrel{\text{def}}{=} \{(x,y) \mid (x,y) \in S, x < y\}$$
$$S_= \stackrel{\text{def}}{=} \{x \mid (x,x) \in S\}$$
$$S_> \stackrel{\text{def}}{=} \{(y,x) \mid (x,y) \in S, x > y\}$$

and rewrite the query as:

$$Q_2' = \exists x \exists y (S_<(x,y) \wedge S_>(x,y)) \vee \exists x S_=(x)$$

The rewritten query Q_2' is ranked (using the variable order x, y) and is equivalent to Q_2, hence $\mathbf{P}(Q_2) = \mathbf{P}(Q_2')$ and

$$\mathbf{P}(Q_2') = 1 - (1 - \mathbf{P}(\exists x \exists y (S_<(x,y) \wedge S_>(x,y))))(1 - \mathbf{P}(\exists x S_=(x)))$$

The reader may continue the calculation by noting that x is a separator variable in $\exists y(S_<(x,y) \wedge S_>(x,y))$.

We end this section on some historical notes on checking independence for FO formulas. For any atom A, denote $\mathrm{ground}(A)$ the set of its groundings. The term "shattered" was introduced by de Salvo Braz et al. [2005] who called a sentence *shattered* if for any pair of atoms A_1, A_2, their sets of grounded atoms are either identical or disjoint, $\mathrm{ground}(A_1) = \mathrm{ground}(A_2)$ or $\mathrm{ground}(A_1) \cap \mathrm{ground}(A_2) = \emptyset$. Notice that our use of the term "shattered" in Definition 4.4 is different. If for all atoms A_1 in Q_1 and A_2 in Q_2 we have $\mathrm{ground}(A_1) \cap \mathrm{ground}(A_2) = \emptyset$, then clearly Q_1, Q_2 are independent. Miklau and Suciu [2007] showed that the converse fails in general: there exists sentences Q_1, Q_2 that are independent yet have common groundings. For example, consider $Q_1 = \exists x \exists y \exists z \exists u (R(x,y,z,z,u) \wedge R(x,x,x,y,y))$ and $Q_2 = R(a,a,b,b,c)$, where a, b, c are three distinct constants. On one hand the tuple $t = R(a,a,b,b,c)$ occurs as grounding for both queries (since $t \in \mathrm{ground}(R(x,y,z,z,u))$). On the other hand, we claim that Q_1, Q_2 are independent. Indeed, the only term in the DNF lineage $F_{Q_1,D}$ that contains the tuple t is $t = R(a,a,b,b,c) \wedge R(a,a,a,a,a)$, and it is absorbed by the term $R(a,a,a,a,a) \wedge R(a,a,a,a,a)$.

The Dichotomy Theroem

Recall that we assume all input relations to be tuple-independent.

Theorem 4.5. Let Q be any query in $FO^{\neg un,\exists,\vee,\wedge}$. Assume w.l.o.g. that Q is shattered, and ranked. Then:

- If the lifted inference rules succeed on Q then computing $\mathbf{P}(Q)$ is in polynomial time in the size of the input database; in this case we say that Q is liftable.

- If the lifted inference rules fail on Q, then computing $\mathbf{P}(Q)$ is provably #P-hard in the size of the input database; in this case we say that Q is not liftable.

The proof of the first item is straightforward. The proof of the second item is significantly harder, and can be found in Dalvi and Suciu [2012].

We illustrate a simple liftable query that requires the use of the inclusion/exclusion formula. Consider the following conjunctive query:

$$Q = \exists x \exists y \exists u \exists v (R(x) \wedge S(x,y) \wedge T(u) \wedge S(u,v))$$

The query can be written as a conjunction of two sentence $Q = Q_1 \wedge Q_2$ where:

$$Q_1 = \exists x \exists y R(x) \wedge S(x,y) \qquad Q_2 = \exists u \exists v T(u) \wedge S(u,v)$$

Notice that, although the query is hierarchical, Theorem 4.2 does not apply, because the query has self-joins. In particular, we cannot apply the Join rule: $\mathbf{P}(Q_1 \wedge Q_2) \neq \mathbf{P}(Q_1)\mathbf{P}(Q_2)$ because Q_1, Q_2 are dependent, since they share the relation symbol S. Instead we apply inclusion/exclusion, and write $\mathbf{P}(Q) = \mathbf{P}(Q_1) + \mathbf{P}(Q_2) - \mathbf{P}(Q_1 \vee Q_2)$. The probabilities of Q_1 and Q_2 can be computed easily since both are hierarchical queries without self-joins. We discuss how to compute the probability of $Q_1 \vee Q_2$. We first rename u with x in Q_2, then use the fact that the existential quantifier commutes with disjunction, therefore:

$$Q_1 \vee Q_2 = \exists x [(R(x) \wedge \exists y S(x,y)) \vee (T(x) \wedge \exists v S(x,v))]$$

Now x is a separator variable, because it occurs in all atoms, and it occurs on the first position of both S-atoms. Therefore, we apply the $\exists$ rule, then simplify the first-order formula by applying the distributivity law

$$\begin{aligned}
\mathbf{P}(Q_1 \vee Q_2) &= 1 - \prod_{a \in D} (1 - \mathbf{P}((R(a) \wedge \exists y S(a,y)) \vee (T(a) \wedge \exists v S(a,v)))) \\
&= 1 - \prod_{a \in D} (1 - \mathbf{P}((R(a) \vee T(a)) \wedge \exists y S(a,y))) \\
&= 1 - \prod_{a \in D} (1 - \mathbf{P}((R(a) \vee T(a))) \mathbf{P}(\exists y S(a,y)))
\end{aligned}$$

and from here it is easy to compute the remaining probability expressions.

Which queries are not liftable? The dichotomy theorem gives us a procedure to check liftability, namely apply the rules and see if they fail, but this does not given any intuition of how the non-liftable queries look like. In general, no other characterization of liftable/non-liftable queries is known, and the query complexity for checking whether the query is in PTIME or #P-hard is open. A good representative list (but still incomplete) of non-liftable queries are the queries H_k below:

$$
\begin{aligned}
H_0 =& \exists x_0 \exists y_0 R(x_0) \wedge S(x_0, y_0) \wedge T(y_0) \\
H_1 =& \exists x_0 \exists y_0 R(x_0) \wedge S_1(x_0, y_0) \vee \exists x_1 \exists y_1 S_1(x_1, y_1) \wedge T(y_1) \\
H_2 =& \exists x_0 \exists y_0 R(x_0) \wedge S_1(x_0, y_0) \vee \exists x_1 \exists y_1 S_1(x_1, y_1) \wedge S_2(x_1, y_1) \\
& \vee \exists x_2 \exists y_2 S_2(x_2, y_2) \wedge T(y_2) \\
H_3 =& \exists x_0 \exists y_0 R(x_0) \wedge S_1(x_0, y_0) \vee \exists x_1 \exists y_1 S_1(x_1, y_1) \wedge S_2(x_1, y_1) \\
& \vee \exists x_2 \exists y_2 S_2(x_2, y_2) \wedge S_3(x_2, y_2) \vee \exists x_3 \exists y_3 S_3(x_3, y_3) \wedge T(y_3) \\
H_4 =& \cdots
\end{aligned}
$$

It is easy to check (and we invite the reader to verify) that no lifted inference rule applies to H_k, for any $k \geq 0$. In fact, Dalvi and Suciu [2012] prove that for every $k \geq 0$, the probability computation problem $\mathbf{P}(H_k)$ is #P-hard in the size of the database. Notice that all queries H_k for $k \geq 1$ are hierarchical (H_0 is, of course, non-hierarchical). We will return to these queries in Chapter 5.

4.5 Negation

In this section we examine queries beyond $FO^{\neg un,\exists,\vee,\wedge}$ or $FO^{\neg un,\forall,\vee,\wedge}$. One can apply the lifted inference rules to any sentence (Boolean query) in FO, and when they succeed, then the query is computable in polynomial time. The question is what can we say when the rules fail on some query Q: is this because Q is hard, or because the rules are insufficient to handle such queries? We note that checking independence is undecidable in FO, and the same can be shown for checking

whether a variable is a separator variable, so it only makes sense to ask this question for fragments of FO where the independence test is decidable. Queries in $FO^{\neg un,\exists,\vee,\wedge}$ and $FO^{\neg un,\forall,\vee,\wedge}$ admit a unique canonical form and, hence, independence is decidable; here we examine other such languages.

Two extensions of unate queries have been considered in the literature. Fink and Olteanu [2014] studied *non-repeating relational algebra* expressions. These are expressions in the relational algebra, i.e. using the operators selection, projection, join, union, and difference, where no relational symbol is allowed to occur more than once. Such expressions are still unate (since every symbol occurs at most once), but the unrestricted use of the difference operator results in arbitrary combinations of $\exists$ and $\forall$ quantifiers. They established the following result:

Theorem 4.6. Let Q be a non-repeating relational algebra expression.

- If Q is hierarchical, then $\mathbf{P}(Q)$ can be computed in polynomial time.
- If Q is non hierarchical, then $\mathbf{P}(Q)$ is #P-hard in the size of the database.

The formal definition of a hierarchical query is an extension of that for first-order sentences, but it is rather subtle because of the need to define precisely what a variable is, and when does it "occur" in a relation name. For example, in an expression like $S - R \times T$, assuming S binary and R, T are unary, the first attribute of S "occurs" in R, and the second attribute "occurs" in T. The theorem gives a dichotomy for non-repeating relational algebra expressions; we refer the reader to Fink and Olteanu [2014] for details.

A second extension was to $FO^{\neg,\forall,\vee,\wedge}$ and $FO^{\neg,\exists,\vee,\wedge}$, and is discussed by Gribkoff et al. [2014a]. Unlike unate formulas, a relational symbol may occur both positively and negatively; the only restriction being that the query expression can use only one type of quantifier. While a full characterization of the complexity remains open for these two languages, the authors have shown that the rules need to be extended with resolution in order to be complete. To illustrate, consider

the following constraint:

$$\Gamma = \forall x \forall y (R(x) \vee S(x,y)) \wedge (\neg S(x,y) \vee T(y))$$

None of the lifted inference rules apply to this query. In fact, if we remove the negation operator, then we obtain a constraint that is the dual of H_1, which we have seen is #P-hard. This means that any inference rules that compute this query must use in an essential way *both* atoms $S(x,y)$ and $\neg S(x,y)$.

Resolution is the logical inference method that takes two clauses of the form $(A_1 \vee A_2 \vee \cdots \vee L)$, $(\neg L \vee B_1 \vee B_2 \vee \ldots)$ and adds the clause $(A_1 \vee A_2 \vee \cdots \vee B_1 \vee B_2 \vee \cdots)$. In other words, writing $A = \neg(A_1 \wedge A_2 \cdots)$ and $B = (B_1 \vee B_2 \vee \cdots)$, resolution takes $(A \Rightarrow L)$ and $(L \Rightarrow B)$ and produces $(A \Rightarrow B)$.

Resolution may help the lifted inference rules get unstuck. Continuing our example, after applying resolution to Γ we obtain the new clause $(R(x) \vee T(y))$:

$$\begin{aligned}
\Gamma =& \forall x \forall y (R(x) \vee S(x,y)) \wedge (\neg S(x,y) \vee T(y)) \wedge (R(x) \vee T(y)) \\
=& \forall x \forall y (R(x) \vee S(x,y)) \wedge (\neg S(x,y) \vee T(y)) \\
& \wedge R(x) \bigvee \forall x \forall y (R(x) \vee S(x,y)) \wedge (\neg S(x,y) \vee T(y)) \wedge T(y) \\
=& \forall x \forall y [(\neg S(x,y) \vee T(y)) \wedge R(x)] \\
& \bigvee \forall x \forall y [(R(x) \vee S(x,y)) \wedge T(y)] \\
=& \forall x \forall y (\neg S(x,y) \vee T(y)) \wedge \forall x R(x) \\
& \bigvee \forall x \forall y (R(x) \vee S(x,y)) \wedge \forall y T(y)
\end{aligned}$$

Hence,

$$\begin{aligned}
\mathbf{P}(\Gamma) = & \mathbf{P}(\forall x \forall y (\neg S(x,y) \vee T(y)) \wedge \forall x R(x)) \\
& + \mathbf{P}(\forall x \forall y (R(x) \vee S(x,y)) \wedge \forall y T(y)) \\
& - \mathbf{P}(\forall x \forall y (\neg S(x,y) \vee T(y)) \wedge \forall x R(x) \\
& \qquad \wedge \forall x \forall y (R(x) \vee S(x,y)) \wedge \forall y T(y)) \\
= & \mathbf{P}(\forall x \forall y (\neg S(x,y) \vee T(y))) \cdot \mathbf{P}(\forall x R(x)) \\
& + \mathbf{P}(\forall x \forall y (R(x) \vee S(x,y))) \cdot \mathbf{P}(\forall y T(y)) \\
& - \mathbf{P}(\forall x R(x) \wedge \forall y T(y)).
\end{aligned}$$

We applied inclusion/exclusion to compute $\mathbf{P}(\Gamma)$ and used simple equivalences of first-order sentences. The remaining probability expressions can be computed easily using a few lifted inference rules.

We end with a remark on the complexity of applying the lifted inference rules. Most rules have preconditions, e.g. the query must be of a certain pattern, or a variable must be a separator variable, etc. To apply a lifted inference rule to a query Q we must first find an equivalent expression $Q' \equiv Q$ such that Q' satisfies the conditions of some rule. For Unions of Conjunctive Queries, query equivalence is decidable and NP-complete; in fact it suffices to first compute the minimal query Q' equivalent to Q (which is uniquely defined up to isomorphism), then apply the rules to the minimized query (and the same holds for $FO^{\neg un,\exists,\vee,\wedge}$ and $FO^{\neg un,\forall,\vee,\wedge}$). Resolution is just another tool in our toolbox for proving query equivalence, needed for queries that make true use of negation. In general, however, it is undecidable whether to FO queries are equivalent, by Trakhtenbrot's theorem, see Libkin [2004]. This means that it is unlikely to extend the lifted inference rules to the entire FO.

4.6 Symmetric Databases

A symmetric probabilistic relation is a tuple-independent relation where every ground tuple over the domain has the same probability. A symmetric probabilistic database is a database where each relation is a symmetric probabilistic relation. Notice that the probability may differ between relations, but will be the same for all tuples belonging to the same relation.

Symmetric databases were motivated by Markov Logic Networks, where the weights are associated to formulas rather than data. When an MLN is converted to a tuple-independent database, as described in §2.6, then the newly introduced relations A are symmetric.

When the database is symmetric then we can use de Finetti's exchangeability theorem in addition the rules in §4.2. Stated in our setting, the theorem says the following.

Fix a Boolean query Q, and suppose $R_1, R_2, \ldots, R_\ell$ are all unary relations occurring in Q. Suppose the domain has size n. Fix relational instances for all ℓ unary relations, $R_i \subseteq [n]$ for $i \in [\ell]$, and, for each subset $S \subseteq [\ell]$ define:

$$C_S = \bigcap_{i \in S} R_i \cap \bigcap_{i \notin S} \bar{R}_i$$

We call the 2^ℓ unary relations C_S the *cells* defined by the relations R_i. Consider the conditional probability $\mathbf{P}(Q|R_1, \ldots, R_\ell)$. By de Finetti's exchangeability theorem, this probability depends only on the cardinalities of the cells, denoted $k_S = |C_S|$, thus we obtain:

$$\mathbf{P}(Q|R_1, \ldots, R_\ell) = \mathbf{P}(Q|(k_S)_{S \subseteq [\ell]}) \tag{4.1}$$

We invite the reader to pause and reflect about this statement. In particular, note that the statement is false if one of the relations R_i is not unary: for example if R_i is binary, then a particular instance of R_i represents a graph, and the probability of Q depends not just on the number of edges k_i, but also on the entire degree sequence, and on much more [Grohe, 2017, pp.105]. Thus, de Finetti's theorem allows us to prove Equation 4.1 only when all relations R_i are unary. From here we derive:

$$\mathbf{P}(Q) = \sum_{k_S = 0, n, S \subseteq [\ell]} \prod_{S \subseteq [\ell]} \binom{n}{k_S} \prod_{i \in [\ell]} p_i^{k_{\{i\}}} (1 - p_i)^{n - k_{\{i\}}} \mathbf{P}(Q|(k_S)_{S \subseteq [\ell]})$$

where p_i is the probability of a tuple in R_i (the same for all tuples in R_i). Since the number of relations is constant, so is the number of cells, and the sum has only polynomially many terms, hence $\mathbf{P}(Q)$ is reduced to the problem of computing $\mathbf{P}(Q|(k_S)_{S \subseteq [\ell]})$ forall $k_1, k_2, \ldots$ Sometimes the latter is computable in PTIME, even if computing $\mathbf{P}(Q)$ is #P-hard over general (asymmetric) databases.

Example 4.4. Consider $H_0 = \forall x \forall y (R(x) \vee S(x, y) \vee T(y))$. Consider a symmetric database over a domain of size n, where every tuple in R has probability p_R, and similarly tuples in S, T have probabilities p_S, p_T respectively. Fix two relation instances $R, T \subseteq [n]$, denote k_R, k_T

their cardinalities. Then

$$P(H_0|k_R, k_T) = p_S^{n^2-k_Rk_T}$$

$$P(H_0) = \sum_{k_R,k_T=0,n} \binom{n}{k_R}\binom{n}{k_T} p_R^{k_R} \cdot (1-p_R)^{n-k_R} \cdot p_T^{k_T}(1-p_T)^{n-k_T} \cdot p_S^{n^2-k_Rk_T}$$

The first line holds because the clause $(R(x) \vee S(x,y) \vee T(y))$ is already satisfied by the k_Rk_T tuples $x \in R, y \in T$, thus S must contain all remaining $n^2 - k_Rk_T$ tuples. In this simple example the conditional probability depends only on the cardinalities k_R, k_T, and there was no need to consider all four cells $R \cap T, R \cap \bar{T}, \bar{R} \cap T, \bar{R} \cap \bar{T}$.

The language FO^k consists of all FO sentences restricted to k logical variables, see Libkin [2004]. For example, the query H_0 above is in FO^2. For another example, $\exists x \exists y \exists z \exists u(R(x,y) \wedge S(y,z) \wedge T(z,u))$ uses four variables, but is equivalent to $\exists x \exists y R(x,y) \wedge (\exists x S(y,x) \wedge (\exists y T(x,y)))$, hence it is in FO^2. Van den Broeck et al. [2014] prove that every query in FO^2 can be computed in PTIME over symmetric databases. For example, all queries H_k described earlier can be computed in PTIME over symmetric databases. The proof uses, in essence, de Finetti's exchangeability theorem, Equation 4.1 above, then shows that if all remaining relations are binary, then the conditional probability can be computed in PTIME. We refer to Niepert and Van den Broeck [2014] for a more detailed discussion of finite exchangeability as it applies to lifted inference.

Clearly, if a query is liftable, according to the rules in §4.2, then it is also computable in PTIME on symmetric databases. The result by Van den Broeck et al. [2014] shows that some #P-hard queries can be computed in PTIME if the input database is restricted to be symmetric, by applying de Finetti's theorem to the unary relations. This raises two questions. Are all queries in FO be computable in PTIME over symmetric databases? And is de Finetti's theorem on unary relations the only new rule that we need over symmetric databases?

Gribkoff et al. [2014a] answer both questions negatively. First, they prove that there exists a sentence in FO^3 for which computing the

probability on a symmetric database is #P_1-complete. They also prove that there exists a conjunctive query whose complexity over symmetric databases is #P_1-complete. The class #P_1 consists of all problems in #P over a unary alphabet, and is a complexity class that is difficult to study. Only a few hard problems are known for this class, and none is "natural"; the hard queries described in Gribkoff et al. [2014a] are not "natural". For many natural queries, their complexity over symmetric databases is open, for example we do not know the complexity of the query $\exists x \exists y \exists z (R(x, y) \wedge S(y, z) \wedge T(z, x))$ over symmetric databases.

Second, Gribkoff et al. [2014a] show that the following query can be computed in PTIME over symmetric databases:

$$\forall x \forall y \forall u \forall v (S(x, y) \vee \neg S(u, y) \vee S(u, v) \vee \neg S(x, v))$$

DeFinetti's theorem cannot be applied since there are no unary symbols. Instead, the algorithm described by Gribkoff et al. [2014a] uses dynamic programming over the domain $[n]$. Kazemi et al. [2016] describe an alternative algorithm that performs recursion over the domain size within the query plan. The complexity of this query over asymmetric databases is open to date.

4.7 Extensions

The positive results discussed so far are restricted to databases that are tuple-independent. The negative results (#P-hardness) hold only if one makes no restriction on the tuple-independent database. We summarize here several extensions that have been studied in the literature.

4.7.1 Block-Independent-Disjoint Databases

Dalvi and Suciu [2007a] consider the complexity of query evaluation on Block-Independent-Disjoint databases. They prove a dichotomy into #P-hard and PTIME for conjunctive queries without self-joins. For example consider the following three queries:

$$q_1 = R(\underline{x}), S(\underline{x, y}), T(\underline{y}), U(\underline{u}, y), V(\underline{a}, u)$$
$$q_2 = V(\underline{x}, y), T(\underline{y})$$
$$q_3 = V(\underline{x}, y), W(x, \underline{y})$$

Every underlined attribute represents the key in a possible worlds of that table. Thus, $R(\underline{x})$ is a tuple-independent table, while $U(\underline{u}, y)$ has blocks defined by grouping on the variable u (and a possible world includes at most one tuple from each u-group). In the first query, a represents a constant. Recall that a *disjoint project* operator assumes that all duplicates being eliminated are exclusive probabilistic events, hence their probabilities can be added up. Using this operator, the probability of q_1 can be computed in PTIME, as illustrated below:

$$\begin{aligned}
\mathbf{P}(q) &= \sum_{b\in D} \mathbf{P}(R(\underline{x}), S(\underline{x,y}), T(\underline{y}), U(\underline{b}, y), V(\underline{a}, b)) \\
&= \sum_{b\in D} \mathbf{P}(R(\underline{x}), S(\underline{x,y}), T(\underline{y}), U(\underline{b}, y)) \cdot p(V(\underline{a}, b)) \\
&= \sum_{b\in D}\sum_{c\in D} \mathbf{P}(R(\underline{x}), S(\underline{x,c}), T(\underline{c}), U(\underline{b}, c)) \cdot p(V(\underline{a}, b)) \\
&= \sum_{b\in D}\sum_{c\in D} \mathbf{P}(R(\underline{x}), S(\underline{x,c})) \cdot p(T(\underline{c})) \cdot p(U(\underline{b}, c)) \cdot p(V(\underline{a}, b)) \\
&= \sum_{b\in D}\sum_{c\in D} (1 - \prod_{d\in D} (1 - p(R(\underline{d})) \cdot p(S(\underline{d,c})))) \cdot \\
&\qquad p(T(\underline{c})) \cdot p(U(\underline{b}, c)) \cdot p(V(\underline{a}, b))
\end{aligned}$$

The first line corresponds to a disjoint project operator that eliminates the variable u: since all values of u occur with the same key value a in $V(\underline{a}, v)$, their corresponding events are disjoint, hence the probabilities can be added up. Next, in each term, the value $u = b$ becomes a key in the relation $U(\underline{u}, y)$, allowing us to do a disjoint projection of the variable y. The rest of the query is treated like a hierarchical query over a tuple-independent database.

On the other hand, both queries q_2 and q_3 are proven to be #P-hard in Dalvi and Suciu [2007a].

The paper establishes a dichotomy for conjunctive queries without self-joins over BID tables into PTIME and #P-hard. Recall that, over tuple-independent databases, a conjunctive query without self-joins is in PTIME iff it is hierarchical, and it is not hard to see that the problem "give a query, is it hierarchical?" is in AC^0. On the other hand, Dalvi and Suciu [2007a] proves that the problem "given a query, is it

computable in PTIME over BID tables?" is PTIME-complete: this rules out a test as simple as the hierarchy test.

The complexity of query evaluation over BID tables is open beyond conjunctive queries without self-joins.

4.7.2 Restricting the Input Database

We consider several restrictions on the database that alter the analysis of lifted query processing.

Functional dependencies Returning to tuple-independent databases, we now restrict the set of possible tuples to satisfy certain functional dependencies. For example, we may declare that x is a key in $S(x, y)$. The relation S is still tuple-independent, but all possible tuples have distinct values of x. The dichotomy theorem for conjunctive queries without self-joins proven by Dalvi and Suciu [2007b] already covered such functional dependencies. A much more elegant and simple technique was described by Olteanu et al. [2009]. The technique is very simple: if $x \rightarrow y$ holds in some relation, then modify the CQ by adding y to all relations that contain x: the complexity of the modified query is the same as that of the original query. For example, if x is a key in $S(x, y)$, then the query $\exists x \exists y R(x), S(x, y), T(y)$ is modified to $\exists x \exists y R(x, y), S(x, y), T(y)$. More precisely, we iterate over the possible tuples $a \in R$, and replace each such tuple with $(a, b) \in R$, where b is the unique value for which $(a, b) \in S$: if no such tuple exists in S then delete a from R. The modified query $R(x, y), S(x, y), T(y)$ is hierarchical, hence in PTIME. Extensions beyond conjunctive queries without self-joins are open.

Databases with mixed probabilistic and deterministic relations In this model we assume that each relation in the database schema is marked as being probabilistic or deterministic (in other words, all tuples have probability $= 1$). For example, if R is deterministic and S, T probabilistic, then $\exists x \exists y R(x), S(x, y), T(y)$ is in PTIME, because it admits the safe plan $\Pi_\emptyset(\Pi_y(R \bowtie_x S) \bowtie_y T)$. But if R, T are probabilistic and S deterministic, then $\exists x \exists y R(x), S(x, y), T(y)$ is #P-hard. The

result by Dalvi and Suciu [2007b] covers such cases but only for conjunctive queries without self-joins. The case of more general queries is open. When the database consists of both symmetric probabilistic relations and asymmetric deterministic relations, then queries that are liftable on symmetric databases remain liftable, as long as the Boolean rank of the deterministic relations is bounded [Van den Broeck and Darwiche, 2013].

Queries over databases with a bounded tree-width Amarilli et al. [2015, 2016] prove that every query in Monadic Second Order logic (MSO) can be evaluated in linear time over tuple-independent probabilistic databases with a bounded tree-width. This result should be contrasted with the Dichotomy in §4.4: in that case we restricted the query and considered arbitrary input databases, while here the query is unrestricted (any query in MSO), while the databases are restricted to have bounded tree-width.

The result is an application of Courcelle [1990]'s theorem, which states that every sentence in MSO can be computed in linear time in the size of the (deterministic) database, if the database is restricted to have a bounded tree-width. Courcelle's theorem consists of translating the MSO query into a tree-automaton, and this translation is non-elementary in the size of the query. Amarilli et al. [2015] shows how to adapt Courcelle's translation to obtain an automaton that computes the query on any possible world, i.e. subset of the input database. The subset is given by adding a bit to each tuple in the database, indicating whether it is present or absent. Thus, the automaton reads tuples from the (tree decomposition) of the database, together with the indicator bit stating whether the tuple is present or not, and computes the query on the possible world corresponding to the tuples where the indicator bit is set to 1. Next, they show how to convert the automaton into a polynomial-time, dynamic programming algorithm that computes the probability of the query on a randomly chosen world.

A natural question is whether there exists richer classes of probabilistic databases for which every FO query is computable in polynomial time. Amarilli et al. [2016] answer this negatively, by showing

that there exists a query in first-order logic that is #P-hard on any class of tuple-independent database instances that has unbounded tree-width.

4.7.3 Extending the Query Language

We review lifted query evaluation for extended query languages.

Queries with Interpreted Predicates Olteanu and Huang [2009] considered CQ with inequality predicates, i.e. $<$. They analyzed the complexity of queries consisting of the Cartesian product of distinct relations and an arbitrary set of inequality predicates, as illustrated by the following two examples:

$$q_1 = \exists x \exists y \exists z \exists u \exists v \exists w R(x), S(y), T(z,u), K(v,w), x < z, y < z, y < v$$
$$q_2 = \exists x \exists y \exists z \exists u \exists v \exists w K(z,u), S(x,y), M(v,w), z < x < u, v < y < w$$

The paper describes two results. The first result describes a class of tractable queries. We say that a query is *max-one* if, for every atom, at most one variable in that atom occurs in an inequality predicate. The paper shows that, for every max-one query, its probability on tuple-independent databases can be computed in PTIME. The proof by Olteanu and Huang [2009] consists of showing how to construct a polynomial-size OBDD (which we define in the next chapter). Here we prove that a max-one query can be compute in polynomial time using dynamic programming. For simplicity, we illustrate the proof on the query q_1 above: the general case follows immediately. Choose any variable that is maximal under the predicates $<$: such a variable must exist, otherwise the inequality predicates form a cycle, and the query is unsatisfiable. For q_1, let's choose z. Suppose w.l.o.g. that the values in the z-column of the relation T are $1, 2, \ldots, n$. Thus, n the largest possible value of the variable z, and we may assume w.l.o.g. that all values of x and y in the database are $< n$ (since we can remove values $\geq n$ without affecting the query). On any possible world, there are two cases: (1) $\exists u T(n,u)$: in that case the query is true iff the following residual query is true, $q_1' = R(x), S(y), K(v,w), y < v$. (2) $\neg\exists u T(n,u)$: in that case q_1 is true iff it is true on the world obtained

by removing all tuples of the form $T(n, -)$ from the database. Thus, denoting $\mathbf{P}_k(-)$ the probability on the subset of the database where all values of z are $\leq k$, and all values of x, y are $< k$, we have:

$$\mathbf{P}_n(q_1) = \mathbf{P}(\exists u T(n,u))\mathbf{P}_n(q_1') + (1 - \mathbf{P}(\exists u T(n,u)))\mathbf{P}_{n-1}(q_1)$$

Each term on the right hand side is either the probability of a simpler query (which is computed similarly), or is $\mathbf{P}_{n-1}(q_1)$, which is over a smaller domain.

Second, Olteanu and Huang [2009] consider queries that are not max-one and describe a large class of #P-hard queries. We refer the reader to Olteanu and Huang [2009] for the rather technical definition of this class, and instead will prove that the query q_2 is #P-hard, by reduction from the query $H_0 = \exists x \exists y R(x), S(x,y), T(y)$. Consider a probabilistic database instance R, S, T, and assume w.l.o.g. that all constants in the database are even numbers. Define the instance K, S, M as follows: the relation S is the same, $K = \{(i-1, i+1) \mid i \in R\}$, $M = \{(j-1, j+1) \mid j \in T\}$. It follows that the probabilities of h_0 and q_2 are equal.

Beyond these two results, the complexity of queries with inequalities is open.

Queries with a HAVING Predicate Ré and Suciu [2009] studied the complexity of conjunctive queries without selfjoins, with GROUP-BY and HAVING predicates, with various aggregate functions. They allow the input to be a BID database. The exact complexity of such a query depends on the choice of the aggregate operator (min, max, exists, count, sum, avg, or count(distinct)), and on the choice of the comparison predicate used for that aggregate ($=, \neq, \geq, >, \leq, <$). For various combinations of these parameters, they proved a trichotomy result: some queries are in PTIME, others are #P-hard by admit efficient approximations (i.e. have an FPTRAS), while others are provably hard to approximate. For example, consider the following Boolean queries:

$$q_1[\text{count(distinct } x) \geq k] = R(x), S(x,y,z)$$
$$q_2[\text{count(distinct } y) \geq k] = R(x), S(x,y,z)$$
$$q_3[\text{count(distinct } y) \geq k] = R(x), S(x,y), T(y)$$

The first query checks if there are at least k distinct values x satisfying the body of the conjunctive query, the second checks of there are at least k distinct values y, and similarly for the third.

The paper proves trichotomy results for various combinations of aggregate functions and comparison operators. In particular, it shows that the query q_1 is in PTIME, q_2 is #P-hard but admits an FPTRAS, while q_3 admits a reduction from `#BIS`, which is the problem: *given a bipartite graph* (X, Y, E)*, compute the fraction of subsets of* $X \times Y$ *that are independent sets*. `#BIS` is a complete problem w.r.t. to approximation-preserving reductions, and thus it is believed to be hard to approximate. In other words, q_3 is believed to be hard to approximate.

The complexity of queries with a HAVING clause is open beyond conjunctive queries without self-joins.

4.7.4 Lifted Inference from the Graphical Models Perspective

Lifted inference in AI and probabilistic databases share a common goal: to exploit relational structure for speeding up inference. A key difference is that the AI community has focused on exploiting the symmetry and exchangeability found in probabilistic graphical model templates [Niepert and Van den Broeck, 2014], not on processing probabilistic data as such. The symmetric weighted model counting problem described in §4.6 unifies these perspectives. Lifted inference for probabilistic graphical models was proposed by Poole [2003], spurring a large amount of research on exact inference [de Salvo Braz et al., 2005, Milch et al., 2008, Sen et al., 2009, Choi et al., 2011, Jha et al., 2010, Gogate and Domingos, 2011, Van den Broeck et al., 2011, Kopp et al., 2015, Kazemi et al., 2016]. These works either develop new inference rules, or speed up an existing set of rules [Kazemi and Poole, 2014, 2016, Taghipour et al., 2013]. Significant attention has also gone to approximate lifting techniques that augment sampling, variational, or message passing algorithms with the ability to exploit symmetry [Singla and Domingos, 2008, Kersting et al., 2009, Niepert, 2012, Van den Broeck et al., 2012, Bui et al., 2013, Venugopal and Gogate, 2014, Jernite et al., 2015, Anand et al., 2017]. The AI community studies domain complexity as a notion separate from data and query complexity. In this context, domain-lifted inference refers to PTIME com-

plexity in the size of the domain, regardless of the graphical model or observed data size [Van den Broeck, 2011]. This community studies the complexity of lifted inference in Jaeger [2000], Jaeger and Van den Broeck [2012], Van den Broeck and Davis [2012], Cozman and Mauá [2015, 2016], as well as the work discussed in §4.6 on the complexity of symmetric weighted model counting.

5

Query Compilation

The goal of *knowledge compilation* is to find a compact representation of a Boolean formula F that can be used to efficiently solve certain hard problems on F [Darwiche and Marquis, 2002]. In this paper, the hard problem of interested is model counting, and therefore we define a *compilation* of a Boolean formula F to be some representation such that the (weighted) model count of F can computed in polynomial time in the size of the representation. Usually, the compilation is some type of circuit that represents F, designed in such a manner that we can do weighted model counting efficiently in the size of the circuit. Huang and Darwiche [2005] have shown that any DPLL-based model counting algorithm can be modified to construct a compilation of the Boolean formula, by constructing a circuit from the execution trace of the algorithm. Many types of circuits have been proposed in the literature, see the monograph by Wegener [2000] and the framework by Darwiche and Marquis [2002]; we review the most popular circuits used for weighted model counting in §5.1.

Jha and Suciu [2011, 2013] defined *Query Compilation* to refer to the compilation of the lineage of a query. The fundamental question in query compilation is: given a fixed query, what is the size of the

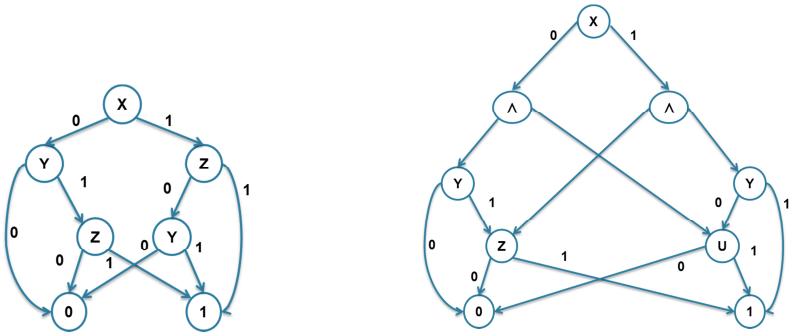
compilation, as a function of the input database? We are interested in this question because it gives us insights into the runtime of inference methods based on grounding. More precisely, consider any algorithm for weighted model counting on a Boolean formula, see §3.4. We can use it to evaluate a query Q on a probabilistic database as follows. First ground the query to obtain a Boolean formula $F_{Q,D}$, then use the algorithm to compute the probability $\mathbf{P}(F_{Q,D})$. The question of interest to us is: for which queries Q is this grounded method guaranteed to run in polynomial time in the size of the database? By the Dichotomy Theorem 4.6, whenever the grounded inference runs in PIME, the query Q is also liftable, hence the answer must be a set of liftable queries. In this chapter we will show that the grounded approach cannot compute efficiently all liftable queries, by proving that some liftable queries have exponential size compilation targets. This shows that the lifted inference methods in Chapter 4 are sometimes exponentially faster that grounded inference.

5.1 Compilation Targets

We briefly review the main compilation targets used in query compilation.

Free Binary Decision Diagrams These circuits were introduced by Lee [1959] and later popularized by Akers Jr. [1978], under the name Binary Decision Diagrams, or Read-Once Branching Programs. They are referred today as Free Binary Decision Diagrams (FBDD) to distinguish from the Orderend Binary Decision Diagrams (discussed next).

Fix a set of Boolean variables $\mathbf{X}$. An *FBDD* is a rooted DAG $\mathcal{F}$, where each internal node u is labeled with a variable $X \in \mathbf{X}$, and has two outgoing edges, labeled 0 and 1; we call its two children the 0-child and the 1-child respectively. Each sink node is labeled either 0 or 1. Moreover, the FBDD is required to satisfy the following condition: for any path from the root node to a sink node, each variable X is read at most once. Figure 5.1a illustrates a simple FBDD.



(a) An FBDD representing Boolean formula $(\neg X)YZ \vee XY \vee XZ$.

(b) A decision-DNNF representing Boolean formula $(\neg X)YZU \vee XYZ \vee XZU$.

Figure 5.1: Illustration of compilation targets from Beame et al. [2017].

The FBDD $\mathcal{F}$ represents a Boolean formula F, defined as follows. Associate to each node u of the FBDD a Boolean formula, F_u, defined inductively as follows: for a 0-sink node, $F_u = 0$, for a 1-sink node, $F_u = 1$, and for any internal node u labeled with a variable X:

$$F_u = (\neg X) \wedge F_{u_0} \vee X \wedge F_{u_1} \tag{5.1}$$

where u_0 and u_1 are its 0-child and 1-child respectively. Then the FBDD denotes the formula $F \stackrel{\text{def}}{=} F_r$, where r is the root of the DAG. Notice that the definition in Equation 5.1 corresponds precisely to a Shannon expansion. In other words, an FBDD encodes a sequence of Shannon expansions.

An equivalent way to define the Boolean F represented by the FBDD $\mathcal{F}$ is as program that computes the Boolean formula, as follows. Let θ be an assignment to the variables $\mathbf{X}$. To compute $\theta(F)$, follow a certain path in the DAG $\mathcal{F}$, as follows. Set the current node to be the root node. If X is the variable labeling the current node, then read its value $\theta(X)$: if $\theta(X) = 0$ then continue with the 0-child, otherwise continue with the 1-child. When reaching a sink node, return $\theta(F)$ as the label of that sink node (0 or 1). We invite the reader to check that the definitions are equivalent.

If $\mathcal{F}$ is an FBDD computing a Boolean formula F, then one can compute the probability of F using dynamic programming on the DAG $\mathcal{F}$. We start by setting $\mathbf{P}(F_u) = 0$ for every 0-sink node u, $\mathbf{P}(F_u) = 1$ for every 1-sink node u, then traverse the DAG bottom-up, and for each node u we set:

$$\mathbf{P}(F_u) = (1 - p(X))\mathbf{P}(F_{u_0}) + p(X)\mathbf{P}(F_{u_1})$$

where u_0, u_1 are its 0-child and 1-child respectively. This takes linear time in the size of the DAG $\mathcal{F}$, since each node u is visited only once.

Given a Boolean formula F with n Boolean variables, our goal is to find a compact FBDD. Once such an FBDD has been constructed, model counting can be performed in linear time in the size of the FBDD. However, there are Boolean formulas F for which any FBDD has size exponential in the number of variables n, as we will illustrate in §5.2.

Ordered Binary Decision Diagrams Fix a variable order Π; more precisely, Π is a permutation on the set $[n]$, where $n = |\mathbf{X}|$ is the number of Boolean variables. A Π-*Ordered Binary Decision Diagram* (OBDD) is an FBDD with the property that every path from the root to a sink node reads the variables in the order given by Π: more precisely if it reads X_i before X_j, then $\Pi(i) < \Pi(j)$ [Wegener, 2000, pp.45]. An OBDD is an Π-OBDD for some Π.

OBDDs where introduced by Bryant [1986]. The main motivation for restricting the variable order was to simplify the synthesis of the OBDD. For any Π, there exists a trivial Π-OBDD with $2^n - 1$ internal nodes, which consists of a full binary tree. Each path in the tree reads the variables in the same order $X_{\Pi^{-1}(1)}, X_{\Pi^{-1}(2)}, \ldots$, there are 2^n paths, one for each assignment to the Boolean variables, and the sink nodes are labeled with the value of the Boolean formula for that assignment. We want to reduce the size of this OBDD. Define an equivalence relation on the set of nodes, where u, v are equivalent if they represent the same Boolean formula, $F_u = F_v$. To reduce the size of the OBDD, we merge all equivalent nodes into one: this operation is well defined, because whenever two nodes u, v are equivalent, then their 0-children are also equivalent, and so are their 1-children. The resulting OBDD is

called the reduced OBDD, or the canonical OBDD for the given variable order Π. Reducing an OBDD to obtain the reduced (canonical) OBDD is a process similar to minimizing a deterministic automaton. Notice that checking formula equivalence $F_u = F_v$ is co-NP complete, and therefore reducing an OBDD is not an effective procedure. While for theoretical analysis we always assume that a Π-OBDD is reduced, in practice systems use some heuristics with false negatives for the equivalence test, resulting in an OBDD that is not fully reduced.

If every path reads all n variables (in the order Π) then we call the OBDD *complete*. The OBDD can then be partitioned into layers, where layer i reads the variable $X_{\Pi^{-1}(i)}$ and both its children belong to layer $i+1$. Notice that a canonical OBDD is not necessarily layered, since edges may skip layers. Every OBDD can be converted into a complete OBDD by introducing dummy test nodes at skipped layers, with at most a factor n increase in the size. The *width* of a complete OBDD is the largest number of nodes at each layer. The number of nodes in the OBDD is $\leq nw$, where n is the number of Boolean variables and w is the width.

An important property of complete OBDDs, which we will use for query compilation, is that one can synthesize an OBDD for a Boolean formula $F = F_1$ op F_2 from OBDDs for its sub-formulas, where op is one of $\vee$ or $\wedge$. Let $\mathcal{F}_1, \mathcal{F}_2$ be complete Π-OBDDs computing F_1, F_2, and let w_1, w_2 be their widths. We construct a Π-OBDD $\mathcal{F}$ for $F = F_1$ op F_2, with a width at most $w_1 w_2$, as follows. The nodes of $\mathcal{F}$ are all pairs (u, v) where u and v are two nodes at the same layer i of $\mathcal{F}_1$ and $\mathcal{F}_2$ respectively. The 0-child and 1-child of (u, v) are (u_0, v_0) and (u_1, v_1) respectively, where u_0, u_1 are the children of u in $\mathcal{F}_1$ and v_0, v_1 are the children of v in $\mathcal{F}_2$. Since both $\mathcal{F}_1$ and $\mathcal{F}_2$ are complete, all nodes u_0, u_1, v_0, v_1 are at the same layer $i+1$, hence the construction is well defined. The root of $\mathcal{F}$ is (r_1, r_2), where r_1, r_2 are the roots of $\mathcal{F}_1, \mathcal{F}_2$, and the sink nodes of $\mathcal{F}$ will be pairs (u, v) where u, v are sink nodes of $\mathcal{F}_1$ or $\mathcal{F}_2$ respectively, and its label is u op v. For example, if we want to compute $F_1 \vee F_2$, then we obtain 4 kind of sink nodes: $(0,0), (0,1), (1,0), (1,1)$. The first becomes a 0-sink in $\mathcal{F}$, the last three become 1-sinks.

Given a Boolean formula F, our goal is to find a variable order Π for which the reduced Π-OBDD is smallest. In general, finding the optimal order Π is NP-complete, but, as we will show, when the formula is the lineage of a query then an efficient order can easily be found, or determine that none exists.

Read-Once Formulas A read-once Boolean expression is an expression where each Boolean variable occurs only once. In this paper we restrict read-once expressions to use only the operators $\wedge, \vee, \neg$; we do not allow `xor` or $\Leftrightarrow$. Like the previous compilation targets, read-once expressions have the property that their probability can be computed in linear time in the size of the formula, using two simple rules $\mathbf{P}(F_1 \wedge F_2) = \mathbf{P}(F_1)\mathbf{P}(F_2)$ and $\mathbf{P}(\neg F) = 1 - \mathbf{P}(F)$ (we use de Morgan's laws to we express $\vee$ in terms of $\neg$ and $\wedge$), and this justifies our interest in read-once expressions. However, they do not strictly speaking form a compilation target, since not every Boolean formula F can be written as a read-once formula. When F can be written as a read-once expression, then we call it a *read-once formula*.

Every read-once formula F with n variables admits an OBDD with $\leq n$ internal nodes. Indeed, let Π be the order in the variables are listed on the leaves of the read-once expression for F. Then the Π-OBDD for F can be obtained inductively on the structure of F. If $F = F_1 \wedge F_2$ then the order Π is the concatenation of the orders of the variables in F_1 and F_2 (which are disjoint sets, by assumption). The OBDD for F is obtained by re-routing the 1-sink node of F_1 to the root node of F_2. Contrast this to the earlier construction for synthesizing and OBDD for $F_1 \wedge F_2$ from two Π-OBBDD's for F_1, F_2; here F_1, F_2 have disjoint sets of variables, and the construction is much simpler. Similarly, the OBDD for $F_1 \vee F_2$ is obtained by re-routing the 0-sink node of the OBDD for F_1 to the root node of F_2. Finally, an OBDD for $\neg F_1$ is obtained from an OBDD for F_1 by switching the 0- and 1-sink nodes. Thus, the OBDD for any read-once formula has linear size. This construction breaks if the read-once formula uses `xor` or $\Leftrightarrow$, see Wegener [2000].

DNNFs and Variants Darwiche [2001] introduced Deterministic Decomposable Negation Normal Forms, or d-DNNF's. A d-DNNF is a rooted DAG where the leaves are labeled with Boolean variables, and the internal nodes are labeled with $\wedge, \vee, \neg$. There are three restrictions on the circuit. First all negations are pushed down to the leaves, hence the term *negation normal form*, NNF. Second, the subtrees of a $\wedge$ node have disjoint sets of variables, in which case we say that the node is *decomposable*, hence the D in DNNF. Finally, the children of a $\vee$ node define mutually exclusive expressions: if u, v are two such children denoting the formulas F_u, F_v respectively, then $F_u \wedge F_v \equiv$ false: then the node is called *deterministic*, hence the d in d-DNNF. Given a d-DNNF for a Boolean formula F, one can compute the probability of F in linear time in the size of the d-DNNF. Notice that a d-DNNF is not effectively checkable: checking if a $\vee$ node is deterministic is coNP-hard. Huang and Darwiche [2005] introduced a restricted class, called *decision-DNNF*, where each $\vee$ node must be of the form $[(\neg X) \wedge F] \vee [X \wedge G]$. In effect, a $\vee$ nodes is similar to a decision node of an FBDD: it checks the variable X and returns either the left child F or the right child G. A Decision-DNNF can be checked effectively. Although Decision-DNNF's were defined as a restriction on d-DNNFs, we prefer to define them as an extension of FBDDs.

We define a Decision-DNNF to be an FBDD extended with a new kind of node, $\wedge$, such that for every $\wedge$-node with children u and v, the subtrees rooted at u and v have disjoint sets of variables. Figure 5.1b illustrates a simple Decision-DNNF. The probability of a Boolean formula is computable in linear time in the size of the Decision-DNNF, using simple dynamic programming. For a decision node u with children u_0, u_1:

$$F_u = (1 - p(X)) \cdot \mathbf{P}(F_{u_0}) + p(X) \cdot \mathbf{P}(F_{u_1}).$$

For an AND node u with children v, w:

$$F_u = \mathbf{P}(F_v) \cdot \mathbf{P}(F_w).$$

Finally, Darwiche [2011] introduces another restriction of d-DNNF's, called *Sentential Decision Diagrams*, SDD, which impose two restrictions on d-DNNFs. First, every internal node is of the form:

$(G_0 \wedge F_0) \vee (G_1 \wedge F_1) \vee \cdots$ such that all G_i are mutually exclusive and exhaustive. That is, $G_i \wedge G_j \equiv$ false, for all $i \neq j$ and $\bigvee_i G_i \equiv$ true. Hence, the node has $2k$ children, corresponding to the expressions G_0, F_0, G_1, F_1, etc. Second, variables are traversed in a fixed order, in the following sense. Let Π be a binary tree whose leaves are labeled with the variables $\mathbf{X}$ of the Boolean expression: Π is called a v-tree. An SDD is called a Π-SDD if, either:

1. Π is a single leaf node representing the variable X and the SDD is a literal of X or a constant, or,

2. denoting Π_1, Π_2 the two subtrees of Π's root node, the children $G_0, G_1, \ldots$ of the SDD's root node are Π_1-SDDs, and the children $F_0, F_1, \ldots$ are Π_2-SDDs.

Any SDD is required to be a Π-SDD for some v-tree Π. SDDs naturally generalize OBDDs: an OBDD is a special case of an SDD where each internal node has the form $(\neg X) \wedge F_0 \vee X \wedge F_1$, and the v-tree Π is a right-deep linear tree, corresponding to a linear order of the variables.

The trace of a DPLL Algorithm Recall from §3.4 that today's exact model counting algorithm are based on the DPLL family of algorithm. Huang and Darwiche [2005] observed that the trace of any DPLL algorithm is a compilation target. More precisely:

- The trace of the basic DPLL algorithm is a complete binary decision diagram (complete binary tree).

- The trace of a DPLL algorithm with caching and static variable order is an OBDD.

- The trace of a DPLL algorithm with caching and dynamic variable order is an FBDD.

- The trace of a DPLL algorithm with caching, dynamic variable order, and components, is a Decision-DNNF.

Indeed, a Shannon expansion step corresponds to constructing a decision node. Caching, and reusing values from the cache, corresponds to sharing nodes in a DAG. Changing dynamically the variable to expand corresponds to different variable orders in the FBDD. And, finally, the "components" extension of the DPLL algorithm corresponds to a decomposable $\wedge$ in a Decision-DNNF.

5.2 Compiling UCQ

Query compilation for some query Q means first computing the lineage $F_{Q,n}$ of the query on the domain $[n]$, then compiling $F_{Q,n}$ into one of the compilation targets described in the previous section. The main problem that we study is the size of the resulting compilation, as a function of the domain size n. If this size is large, e.g. exponential in n, then any DPLL-based algorithm whose trace is that type of compilation will also run in exponential time. If the size is small, then in most cases it turns out that we can also design a specialized DPLL algorithm to compute the query with that running time.

We will present several lower and upper bounds for the compilation size. Most of these results were presented for Unions of Conjunctive Queries (UCQ), but they carry over immediately to Unate First Order Logic with a single type of quantifiers ($\exists$ or $\forall$), similarly to Chapter 4. To keep the discussion simple, we present these results in the context of Unions of Conjunctive Queries. Recall that we denoted $FO^{\exists,\vee,\wedge}$ the positive, existential fragment of FO. $FO^{\exists,\vee,\wedge}$ is equivalent to Boolean UCQ queries, which are usually written as a disjunction of conjunctive queries $\bigvee_m Q_m$. The conversion from $FO^{\exists,\vee,\wedge}$ to an expression of the form $\bigvee_m Q_m$ may lead to an exponential blowup in size, but this is of no concern to us, since we only study the data complexity, and for that purpose the query is treated as a constant. In this chapter we will use $FO^{\exists,\vee,\wedge}$ and UCQ interchangeably, with some abuse.

5.2.1 Query Compilation to Read-Once Formulas

Recall that we defined hierarchical queries in §4.3. We give here an alternative, equivalent definition [Suciu et al., 2011, pp.72]. An expression in $FO^{\exists,\vee,\wedge}$ is called hierarchical if for any existentially quantified sub-expression $\exists x Q$, the variable x occurs in all atoms in Q. A UCQ query is called hierarchical if it is equivalent to a hierarchical expressions. The reader may check that, for $FO^{\exists,\vee,\wedge}$, this definition is equivalent[1] to Def. 4.3. For example, we have argued in §4.3 that $\exists x \exists y (R(x) \wedge S(x,y))$ is hierarchical: it is not a hierarchical expression according to our new definition, but it is equivalent to the hierarchical expression $\exists x (R(x) \wedge \exists y S(x,y))$. An expression in $FO^{\exists,\vee,\wedge}$ is called *non-repeating* if every relational symbol occurs at most once.

The following gives a complete characterization of UCQ queries that have read-once lineages:

Theorem 5.1. Jha and Suciu [2013] Let Q be a UCQ query. Then the following conditions are equivalent.

- Q is equivalent to an expression that is both hierarchical and non-repeating.
- For every $n \geq 0$, the lineage $F_{Q,n}$ is a read-once Boolean formula.

The proof in one direction is straightforward: if Q is an expression that is both hierarchical and non-repeating, then its lineage expression as defined in §3.3 is also read-once: we need the non-repeating property to ensure that $F_{Q_1,n} \vee F_{Q_2,n}$ and $F_{Q_1,n} \wedge F_{Q_2,n}$ are read-once, and we use the hierarchy property to ensure that $\bigvee_{i \in [n]} F_{Q[i/x],n}$ is read-once. The proof in the other direction is rather technical, see Jha and Suciu [2013]: it shows that if the lineage $F_{Q,n}$ over a domain of size $n = \Omega(k^m)$ is read-once, then it can be converted into a hierarchical, non-repeating expression for Q; here m is the maximum arity of all relational symbols, and k the number of variables plus number of relational symbols in Q.

[1] The two definitions do not agree for FO. For example $\exists x \forall y (R(y) \wedge S(x,y))$ is hierarchical according to Def. 4.3, but we cannot push $\exists x$ past $\forall y$.

Recall that the class of conjunctive queries is the same as $FO^{\exists,\wedge}$. Any conjunctive query that is hierarchical by Def. 4.3 admits a hierarchical expression by simply pushing $\exists$-quantifiers down. Moreover, a conjunctive query without self-joins is trivially non-repeating. This implies:

Corollary 5.2. Olteanu and Huang [2008] Let Q be a conjunctive query without self-joins. Then Q is hierarchical iff for every $n \geq 0$, the lineage $F_{Q,n}$ is a read-once Boolean formula.

We illustrate with several examples.

Consider the query expression $Q = \exists x(R(x) \wedge \exists y S(x,y))$, which is both hierarchical and non-repeating. The lineage expression is:

$$R(1) \wedge (S(1,1) \vee S(1,2) \vee \cdots) \vee R(2) \wedge (S(2,1) \vee \cdots) \vee \cdots$$

and is obviously read-once.

Next, we consider two queries that we have seen in §4.4 can be computed in PTIME. We start with:

$$Q = \exists x \exists y(R(x) \wedge S(x,y)) \vee \exists u \exists v(T(u) \wedge S(u,v))$$

Its lineage is always read-once because it is equivalent to the following expression that is both hierarchical and non-repeating:

$$Q = \exists x[(R(x) \vee T(x)) \wedge \exists y S(x,y)]$$

The second query is:

$$Q = \exists x \exists y \exists u \exists v(R(x) \wedge S(x,y) \wedge T(u) \wedge S(u,v))$$

While this query is hierarchical, it cannot be written without repeating the symbol S, hence its lineage is not always read-once: we invite the reader to check that the lineage $F_{Q,3}$ is not read-one by the characterization of read-once Boolean formulas described by Golumbic et al. [2006]. This shows that liftable queries are not restricted to those whose lineage is read-once.

We end this section with a discussion on the subtle requirement that the expression for Q needs to be simultaneously hierarchical and

non-repeating to guarantee read-onceness. It is insufficient to find separately a hierarchical and a read-once expression for Q. For example, consider the query H_1:

$$H_1 = \exists x_0 \exists y_0 R(x_0) \wedge S(x_0, y_0) \vee \exists x_1 \exists y_1 S(x_1, y_1) \wedge T(y_1)$$

The following two equivalent expressions are hierarchical and non-repeating respectively, but none has both properties:

$$\begin{aligned} H_1 \equiv & \exists x_0 (R(x_0) \wedge \exists y_0 S(x_0, y_0)) \vee \exists y_1 (\exists x_1 (S(x_1, y_1)) \vee T(y_1)) \\ H_1 \equiv & \exists x \exists y (S(x, y) \wedge (R(x) \vee T(y))) \end{aligned}$$

Computing the probability of H_1 is #P-hard, so obviously its lineage cannot be read-once. This leads to the question whether such counterexamples are restricted to #P-hard queries. The answer is negative: the following two expressions [Suciu et al., 2011, pp.112] are equivalent, one is hierarchical, the other is non-repeating, the query is liftable (hence computable in polynomial time), yet is lineage is not read-once:

$$\begin{aligned} Q \equiv & \exists x_1 \exists y_1 (A(x_1) \wedge B(x_1) \wedge C(y_1) \wedge F(y_1)) \\ & \vee \exists x_2 \exists y_2 (A(x_2) \wedge D(x_2) \wedge E(y_2) \wedge F(y_2)) \\ Q \equiv & \exists x \exists y [(A(x) \wedge ((B(x) \wedge C(y)) \vee (D(x) \wedge E(y))) \wedge F(y_2))] \end{aligned}$$

We invite the reader to prove that every query over a unary vocabulary is liftable using the inference rules in §4.2; this proves that Q is liftable. To prove that its lineage is not read-once in general, it suffices to check that the primal graph of the lineage $F_{Q,2}$ has an induced P_4 path (see Golumbic et al. [2006]).

5.2.2 Query Compilation to polynomial size OBDDs

UCQ Queries with a polynomial size OBDD can be fully characterized syntactically, using a syntactic property called inversion. Let Q be a UCQ query expression. We assume in this section that the query Q is shattered and ranked, see Def. 4.4; the results in this section apply immediately to arbitrary UCQ, because shattering and ranking (Lemma 4.3) do not affect the lineage.

Consider an $FO^{\exists,\vee,\wedge}$ expression Q, and let A be an atom in Q. The *context* of A is the sequence of existentially quantified variables preceding A, and we denote it with $\exists x_1 \exists x_2 \cdots \exists x_k$. The order matters, i.e. the context consists of the order in which the existential quantifiers were introduced. Notice that if Q is a hierarchical expression, then the atom A must contain all variables in the context.

Definition 5.1. [Suciu et al., 2011, pp.112] A query expression Q in $FO^{\exists,\vee,\wedge}$ is called *inversion-free* if the following holds. For any relational symbol R of arity k there exists a permuation π^R on $[k]$ such that, for every atom A of Q that refers to the symbol R, the context of A consists of exactly k variables, $\exists x_1 \exists x_2 \cdots \exists x_k$, and the atom is $A = R(x_{\pi^R(1)}, x_{\pi^R(2)}, \ldots, x_{\pi^R(k)})$.

A query is inversion-free if it is equivalent to an inversion-free expression.

An inversion-free expression is, in particular, a hierarchical expression; moreover, it requires that atoms referring to the same relational symbol use the variables in their context in the same order. For example consider the following two queries:

$$Q = \exists x_1(R(x_1) \wedge \exists y_1 S(x_1, y_1)) \vee \exists x_2(\exists y_2 S(x_2, y_2) \vee T(x_2))$$
$$H_1 = \exists x_1(R(x_1) \wedge \exists y_1 S(x_1, y_1)) \vee \exists y_2(\exists x_2 S(x_2, y_2) \vee T(y_2))$$

The first query is inversion free, because in both atoms S the variables occur in the same order as the existential quantifiers that introduced them: $\exists x_1 \exists y_1 S(x_1, y_1)$ and $\exists x_2 \exists y_2 S(x_2, y_2)$. The second query is not inversion-free because the variables in the two atoms S use reverse orders relative to the existential quantifiers, i.e. $\exists x_1 \exists y_1 S(x_1, y_1)$ and $\exists y_2 \exists x_2 S(x_2, y_2)$. We cannot swap the quantifier order $\exists y_2 \exists x_2$ to $\exists x_2 \exists y_2$ because the context for $T(y_2)$ consists only of $\exists y_2$. For a more subtle example, consider the query H_2 (introduced in §4.4):

$$\begin{aligned} H_2 =& \exists x_0 R(x_0) \wedge \exists y_0 S_1(x_0, y_0) \vee \exists x_1 \exists y_1 S_1(x_1, y_1) \wedge S_2(x_1, y_1) \\ & \vee \exists y_2(\exists x_2 S_2(x_2, y_2) \wedge T(y_2)) \end{aligned}$$

The first occurrence of S_1 requires x_0 to be introduced before y_0, and therefore in the second occurrence of S_1, x_1 must be introduced before

y_1 (we cannot swap $\exists x_1 \exists y_1$ for that reason). But that conflicts with the order required by the second S_2, which needs y_2 to be introduced before x_2. Hence, H_2 too has an inversion. It is easy to check that all queries H_k, $k \geq 0$ in §4.4 have an inversion. On the other hand, every hierarchical, non-repeating query expression is inversion-free.

We give an equivalent definition of inversion-free queries, perhaps more intuitive. We will assume that Q is a hierarchical query expression. Recall from §4.3 that $at(x)$ denotes the set of atoms that contain the variable x. The *unification graph* of Q is defined as follows. Its nodes consists of pairs of variables (x, y) that occur in a common atom. And there is an undirected edge from (x, y) to (x', y') if there exists two atoms containing x, y and x', y' respectively, such that the two atoms can be unified and their most general unifier sets $x = x'$ and $y = y'$. Then an *inversion* is a path $(x_0, y_0), (x_1, y_1), \ldots, (x_k, y_k)$ where $at(x_0) \supset at(y_0)$ and $at(x_k) \subset at(y_k)$. The length of the inversion is defined as k. (We can assume w.l.o.g. that $at(x_i) = at(y_i)$ for $i = 1, k-1$: otherwise, if for example $at(x_i) \supset at(y_i)$, then we consider the shorter inversion from (x_i, y_i) to (x_k, y_k).) One can check that the query expression Q is inversion-free (as defined earlier) iff it has no inversion (as defined here). For example, every query H_k has an inversion of length k, namely $(x_0, y_0), (x_1, y_1), \cdots, (x_k, y_k)$.

Fix a domain size n, and recall that Tup$([n])$ denotes the set of ground tuples over the domain $[n]$.

Theorem 5.3. Jha and Suciu [2013] Let Q be a shattered and ranked UCQ. Then the following hold:

- If Q is inversion free, then for all n there exists an order Π on Tup$[n]$ such that the reduced Π-OBDD for the lineage $F_{Q,n}$ has width $\leq 2^{|Q|} = O(1)$, and size $O(n^k)$, where k is the maximum arity of any relation in Q.
- If Q is a minimal UCQ (§4.4), and has an inversion of length k, then for all n and for any order Π on Tup$[n]$, the reduced OBDD of lineage $F_{Q,n}$ has size $\Omega(k2^{n/2k})$.

The theorem thus proves a dichotomy of UCQ queries into those whose lineage admits a polynomial size OBDD, and, in fact, linear

in the number of Boolean variables $|\text{Tup}(n)|$, and those for which the OBDD is exponential in the size of the domain.

Proof. We sketch only the proof of the first item, by showing how to convert an inversion-free query into an OBDD. We start from an inversion-free query Q and a domain size n. Recall that the database schema is $\mathbf{R} = (\mathsf{R}_1, \mathsf{R}_2, \ldots, \mathsf{R}_\ell)$. Consider the set $S = \{\mathsf{R}_1, \ldots, \mathsf{R}_\ell\} \cup [n]$ with the total order $\mathsf{R}_1 < \mathsf{R}_2 < \cdots < \mathsf{R}_\ell < 1 < 2 < \cdots < n$. By definition, each relational symbol $\mathsf{R} \in \mathbf{R}$ is associated with a permutation π^{R}, and define $\rho^{\mathsf{R}} \stackrel{\text{def}}{=} (\pi^{\mathsf{R}})^{-1}$. We associate each ground tuple $t = \mathsf{R}(i_1, i_2, \ldots, i_k) \in \text{Tup}([n])$ with the following sequence in S^*: $(i_{\rho(i_1)}, i_{\rho(i_2)}, \ldots, i_{\rho(i_k)}, \mathsf{R})$. Define the order Π on $\text{Tup}([n])$ as the lexicographic order of the corresponding sequences. In other words, if A is an atom in the query, then we order the ground tuples $gr(A)$ by their first existential variable, then by their second, and so on; the fact that the query is inversion-free ensures that there is no conflict in this ordering, i.e. we get the same order for $gr(A')$ where A' is a different atom that refers to the same relational symbol as A. We claim that the width of the Π-OBDD for Q is $w \leq 2^{|Q|} = O(1)$, which implies that the size of the Π-OBDD is $O(|\text{Tup}|) = O(n^k)$. We prove the claim by induction on the sentence Q. If Q is a single ground atom t, then the width of the complete Π-OBDD is 2 (all levels below t must have two nodes to remember if t was true or false). If $Q = Q_1 \wedge Q_2$ or $Q = Q_1 \vee Q_2$ then we first construct complete OBDDs for Q_1, Q_2. Since both Q_1, Q_2 use the same variable order Π, we can use the OBDD synthesis described earlier to derive a complete OBDD for Q, whose width is at most the product of the widths of the two OBDDs. If $Q = \exists x Q_1$, then we first construct n OBDDs G_i for $Q_1[i/x]$, for $i = 1, 2, \ldots, n$. Let $T \stackrel{\text{def}}{=} \text{Tup}[n]$. Partition T into n sets $T = T_1 \cup \cdots \cup T_n$, where T_i consists of all ground atoms $\mathsf{R}(i_1, \ldots, i_k)$ where $\mathsf{R} \in \mathbf{R}$ and $i_{\rho^{\mathsf{R}}(1)} = i$. Then, for each i, the lineage of $Q_1[i/x]$ uses only Boolean variables from T_i. Moreover, the order Π places all tuples in T_i before those in T_j, forall $i < j$. Therefore, we can construct an Π-OBDD for $\bigvee F_{Q_1[i/x],n}$ as follows: take the union of all OBDDs $G_1, \ldots, G_n$, reroute the 0-sink node of G_{i-1} to the root note of G_i. The resulting Π-OBDD is not complete yet, because

the 1-sink node of G_{i-1} stops early, skipping all levels in the OBDDs $G_i, \ldots, G_n$. We complete the OBDD (since we need the OBDD to be complete at each step of the induction) by introducing one new node $w_{i,j}$ for each layer j of each G_i, $i > 1$. The 1-sink node of G_{i-1}, $i < n$ will be re-routed to $w_{i,1}$; both the 0-child and 1-child of $w_{i,j}$ are $w_{i,j+1}$ if j is not the last layer of G_i, or both are $w_{i+1,1}$ if $i < n$, or both are the 1-sink node otherwise. The new nodes increase the width only by 1. □

Thus, inversion-free queries prescribe a simple order on the Boolean variables: simply order the attributes of each relation R according to $(\pi^R)^{-1}$, then order its grounded tuples lexicographically. For example, if the query is $\exists x \exists y R(x) \wedge S(x, y)$ then we order S in row-major order: $S(1,1), S(1,2), \ldots, S(2,1), S(2,2), \ldots$ On the other hand, for the query $\exists x \exists y S(x, y) \wedge T(y)$ we order them in column-major order, $S(1,1), S(2,1), \ldots, S(1,2), S(2,2), \ldots$ If the query has an inversion, then there is a conflict between the different orderings, and we have no good choice for the order Π: an OBDD will be exponentially large.

Beame and Liew [2015] have recently extended Theorem 5.3 from OBDDs to SDDs. The first item of the theorem immediately applies to SDDs, since every OBDD is also an SDD. Beame and Liew [2015] extended the second bullet to SDDs, by showing that, if a query has an inversion, then its SDD is exponentially large. Thus, although SDDs were designed specifically to have increased expressive power over OBDDs, when restricted to Unions of Conjunctive Queries, they are effectively the same. Bova [2016] has shown that SDD can be exponentially more concise than OBDDs on the *generalized hidden bit* class of Boolean formulas. The study of the expressive power of SDDs compared to OBDDs and FBDDs is an active area of research.

5.2.3 Query Compilation to polynomial size FBDDs

There are known examples of UCQ queries whose FBDDs have size polynomial in the input domain, and there are known examples of queries whose FBDDs are exponential. We will illustrate both. The exact separation between them is open.

Consider the following query:

$$Q = \exists x_1 \exists y_1 (R(x_1) \wedge S(x_1, y_1)) \vee \exists x_2 \exists y_2 (S(x_2, y_2) \wedge T(y_2)) \\ \vee \exists x_3 \exists y_3 (R(x_3) \wedge T(y_3))$$

We invite the reader to check that $\mathbf{P}(Q)$ can be computed in PTIME using the lifted inference rules in §4.2. On the other hand, the minimal OBDD for its lineage has exponential size, by Theorem 5.3, because the query has an inversion from (x_1, y_1) to (x_2, y_2). Intuitively, the reason why any OBDD is exponentially large is the conflict between the need to read the variables $S(i, j)$ in row-major order in order to compute $\exists x_1 \exists y_1 (R(x_1) \wedge S(x_1, y_1))$, and the need to read them in column-major order to compute $\exists x_2 \exists y_2 (S(x_1, y_1) \wedge T(y_1))$.

Surprisingly, the query has an FBDD whose size is polynomial in the number of Boolean variables[2]. The FBDD starts by testing $R(1)$. If $R(1) = \texttt{true}$, then the query simplifies to:

$$\begin{aligned} Q|_{R(1)=\texttt{true}} =& \exists x_1 \exists y_1 (R(x_1) \wedge S(x_1, y_1))|_{R(1)=\texttt{true}} \\ & \vee \exists x_2 \exists y_2 (S(x_2, y_2) \wedge T(y_2)) \vee \exists y_3 T(y_3) \\ =& \exists x_1 \exists y_1 (R(x_1) \wedge S(x_1, y_1))|_{R(1)=\texttt{true}} \vee \exists y_3 T(y_3) \end{aligned}$$

because the sentence $\exists x_2 \exists y_2 (S(x_2, y_2) \wedge T(y_2))$ logically implies $\exists y_3 T(y_3)$ and therefore is redundant. Thus, on this branch the residual query is inversion-free, and we can compute it with an OBDD by ordering the atoms S in row-major order. On the branch $R(1) = \texttt{false}$, we test $R(2)$. If $R(2) = \texttt{true}$ then the query simplifies to:

$$\begin{aligned} Q|_{R(1)=\texttt{false},R(2)=\texttt{true}} =& \exists x_1 \exists y_1 (R(x_1) \wedge S(x_1, y_1))|_{R(1)=\texttt{false},R(2)=\texttt{true}} \\ & \vee \exists y_3 T(y_3) \end{aligned}$$

which, again, is inversion-free and can be computed by traversing S in row-major order. Thus, the FBDD will consists of a union of n OBDDs, one for each branch $R(1) = \cdots = R(i-1) = \texttt{false}$, $R(i) = \texttt{true}$, where it computes an inversion-free query. We are left with the branch $R(1) = \cdots = R(n) = \texttt{false}$, and here the residual query is:

$$Q|_{R(1)=\cdots=R(n)=\texttt{false}} = \exists x_2 \exists y_2 (S(x_2, y_2) \wedge T(y_2))$$

[2]The size can be reduced to be linear in the number of Boolean variables.

This, too, is inversion-free, and we can compute it with an OBDD by ordering the atoms S in column-major order. Since each of the $n+1$ OBDDs used in the construction has size $O(n^2)$, the total size of the FBDD is $O(n^3)$. Notice that the FBDD uses different orders for the atoms $S(i,j)$ on different branches: row-major on some branches, and column major on other branches.

Next, we describe an interesting class of queries that have been shown to require exponential size. Recall that H_k is the union of the following conjunctive queries:

$$\begin{aligned} H_{k0} =& \exists x_0 \exists y_0 (R(x_0) \wedge S_1(x_0, y_0)) \\ H_{k1} =& \exists x_1 \exists y_1 (S_1(x_1, y_1) \wedge S_2(x_1, y_1)) \\ & \cdots \\ H_{kk} =& \exists x_k \exists y_k (S_k(x_k, y_k) \wedge T(y_k)) \end{aligned}$$

Theorem 5.4. Beame et al. [2014, 2017] (1) Any FBDD for H_k has $\geq (2^n - 1)/n$ nodes, where n is the size of the domain. (2) Let $F(Z_0, Z_1, \ldots, Z_k)$ be any monotone Boolean formula that depends on all variables Z_i, $i = 0, k$. Then any FBDD for the query $Q = F(H_{k0}, H_{k1}, \ldots, H_{kk})$ has size $2^{\Omega(n)}$.

Recall that the number of Boolean variables in the lineage of H_k is $2n + kn^2 = O(n^2)$; part (1) of the theorem says essentially that any FBDD for H_k has size exponential in the square root of the number of variables.

Part (2) of the theorem is the interesting piece, because it allows us to combine the queries $H_{k0}, \ldots, H_{kk}$ in any ways, as long as we use every sub-query H_{ki}. Notice that, if we don't use some sub-query H_{ki}, then $F(H_{k0}, H_{k1}, \ldots, H_{kk})$ is inversion-free, hence it has an OBDD of linear size, so we must use every sub-query to generate queries Q with exponential-size FBDDs. For example, part (2) immediately implies part (1), since H_k is a $\vee$-combination, $H_k = H_{k0} \vee \cdots \vee H_{kk}$, but H_k is not very interesting since we already know that computing $\mathbf{P}(H_k)$ is #P-hard. An interesting query is:

$$Q_W = (H_{30} \vee H_{32}) \wedge (H_{30} \vee H_{33}) \wedge (H_{31} \vee H_{33}) \tag{5.2}$$

The theorem implies that any FBDD for Q_W is exponential in the size of the domain. However, Q_W can be computed in PTIME using lifted inference, by applying the inclusion/exclusion formula:

$$\begin{aligned}\mathbf{P}(Q_W) =& \mathbf{P}(H_{30} \vee H_{32}) + \mathbf{P}(H_{30} \vee H_{33}) + \mathbf{P}(H_{31} \vee H_{33}) \\ & - \mathbf{P}(H_{30} \vee H_{32} \vee H_{33}) - \mathbf{P}(H_{30} \vee H_{31} \vee H_{33}) \\ & - \mathbf{P}(H_{30} \vee H_{31} \vee H_{32} \vee H_{33}) \\ & + \mathbf{P}(H_{30} \vee H_{31} \vee H_{32} \vee H_{33}) \\ =& \mathbf{P}(H_{30} \vee H_{32}) + \mathbf{P}(H_{30} \vee H_{33}) + \mathbf{P}(H_{31} \vee H_{33}) \\ & - \mathbf{P}(H_{30} \vee H_{32} \vee H_{33}) - \mathbf{P}(H_{30} \vee H_{31} \vee H_{33})\end{aligned}$$

The two terms containing the #P-hard expression $H_3 = H_{30} \vee H_{31} \vee H_{32} \vee H_{33}$ canceled out, and we are left with five inversion-free queries. Each can be computed in PTIME (and, in fact, each has a linear-size OBDD).

5.2.4 Query Compilation to polynomial size Decision-DNNFs

Beame et al. [2013, 2017] have shown a general result proving that Decision-DNNF's are not much more powerful than FBDDs:

Theorem 5.5. Beame et al. [2013, 2017] If G is a Decision-DNNF with N nodes computing a Boolean formula F, then there exists an FBDD G' with at most $N2^{\log^2 N}$ nodes computing the same formula.

In other words, for any Decision-DNNF with N nodes we can construct an equivalent FBDD with $\approx N^{\log N}$ nodes: this expression is called a *quasi polynomial*, because it increases slightly faster than any polynomial, but not exponentially fast.

Recall that Decision-DNNFs and FBDDs are traces of DPLL algorithm with and without components. Then, the theorem implies immediately a lower bounds on the runtime of any DPLL algorithm with components:

Proposition 5.1. Any DPLL-based algorithm for model counting takes $2^{\Omega(n^{1/2})}$ steps when applied to the lineage of the query Q_W in Equation 5.2.

Proof. Suppose the algorithm takes N steps. Then we obtain a Decision-DNNF with N nodes. By Theorem 5.5, we also obtain an FBDD of size $\leq 2^{\log N + \log^2 N}$, which, by Theorem 5.4, is $2^{\Omega(n)}$. Thus, $\log^2 N = \Omega(n)$, proving the claim. □

Thus, for some the query Q_W lifted inference is exponentially faster than grounding followed by a DBPLL-based algorithm. The query Q_W is not unique with this property. Beame et al. [2013, 2017] describe an entire class of queries, namely any query $F(H_{k0}, H_{k1}, \ldots, H_{kk})$ where F is a positive Boolean formula, whose lattice L consisting of all its implicates and `false` has the property $\mu_L(\hat{0}, \hat{1}) = 0$, where μ_L is the Möbius function μ_L on L, and $\hat{0}, \hat{1}$ are the minimal and maximal elements of L (in particular $\hat{0}$ is $H_{k0} \vee \cdots \vee H_{kk}$ and $\hat{1}$ is `false`); we refer the reader to Stanley [1997] for the definition of the lattice-theoretic notions, and to Beame et al. [2013, 2017] for the details of the class of queries separating lifted inference from grounded inference.

5.3 Compilation Beyond UCQ

We briefly review compilation approaches that go beyond UCQ queries on tuple-independent probabilistic databases.

Probabilistic Graphical Models Knowledge compilation is an effective way of performing inference in a variety of probabilistic models. For example, weighted model counting encodings of probabilistic graphical models are compiled into d-DNNF [Chavira and Darwiche, 2005] or SDD [Choi et al., 2013] to simplify the development of solvers, and amortize the cost of inference for multiple online queries over a single offline compilation step.

Probabilistic Logic Programs and Datalog The compilation approach to inference has been particularly popular within probabilistic logic programming. De Raedt et al. [2007] and Riguzzi [2007] compile ProbLog and LPADs into OBDDs. Fierens et al. [2015] propose to compile into the more succinct d-DNNF representation instead. While this

can yield exponentially smaller circuits, d-DNNF compilers require CNF sentences as input, necessitating an initial expensive reduction from logic programs to CNFs in propositional logic. To avoid this intermediate CNF representation, Vlasselaer et al. [2015] augment the iterative semi-naive evaluation strategy for datalog to incrementally compile an SDD circuit. This enables the compilation of significantly larger probabilistic datalog programs. Finally, Renkens et al. [2012, 2014] study the approximate compilation problem: from a probabilistic logic program, how to find a CNF that can easily be compiled into an efficient circuit, and whose weighted model count provides a tight lower bound on the query probability.

First-Order Circuits The compilation approaches discussed so far all start from a first-order description (the query) to compile a propositional logic circuit. Van den Broeck et al. [2011] and Van den Broeck [2013] instead define a first-order d-DNNF circuit language that permits efficient first-order model counting. Statistical relational models such as MLNs and parfactors (§2.7.2) are turned into these circuits through a process of *first-order knowledge compilation*. This compilation process can be thought of as a lifted query plan [Gribkoff et al., 2014b], keeping a trace of the execution of the lifted query evaluation algorithm (§4.2) with additional compilation rules for symmetric databases (§4.6). By grounding a first-order d-DNNF for a given domain, one obtains a classical d-DNNF circuit. The size of the first-order d-DNNF is independent of the domain size, and is compact even for some queries that have no compact classical d-DNNF [Van den Broeck, 2015].

6

Data, Systems, and Applications

To conclude, we list several probabilistic datasets, systems, and applications that embody some of the ideas presented in this survey.

6.1 Probabilistic Data

Several large datasets have been reported in the literature that are annotated with probabilistic values. We describe some of them below, and summarize them in Table 6.1.

NELL The *Never-Ending Language Learning* (NELL) project at CMU described by Carlson et al. [2010], is a research project that aims to learn over time to read the web. Started in 2010, NELL extracts facts from text found in hundreds of millions of web pages, and improves its reading competence over time. The facts, called "beliefs", are records whose main four attributes are (entity, relation, value, confidence); the records have several other attributes storing meta data (the source webpage, the actual string that produced the extraction, etc). Currently, NELL reports 50M tuples (beliefs).

Name	URL	Number of tuples	Notes
NELL	http://rtw.ml.cmu.edu/rtw/	50M	
Probase	https://concept.research.microsoft.com/Home/Download	85M	IsA relations
Knowledge Vault	https://en.wikipedia.org/wiki/Knowledge_Vault	1.6B - 3B	Not publicly available
Yago	http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/	120M	Probabilities are not publicly available
ReVerb	http://reverb.cs.washington.edu/	15M	only $p > 0.9$

Table 6.1: Some probabilistic datasets reported in the literature

Probase Microsoft's Probase reported in Wu et al. [2012], is a universal, general-purpose, probabilistic taxonomy, automatically constructed from a corpus of 1.6 billion web pages. It uses an iterative learning algorithm to extract isA pairs from web text. For example, Probase contains the triples (apple, isA, fruit), (apple, isA, company), (tree, isA, plant), and (steam turbine, isA, plant). The database associates two probabilities to each triple. The first is the *plausibility*, which is the probability that the triple exists, quite similar to a tuple-independent database. Then second is *typicality*, which corresponds to a conditional probability. Typicality is defined in both direction: for a given concept how typical is a given instance, and for a given instance how typical is the concept; each corresponds to a BID table. Probase now forms the Microsoft Concept graph; only a small subset of the data (without probabilities) is available for download.

Knowledge Vault Dong et al. [2014] describe Google's Knowledge Vault project, or KV for short, which constructs automatically a Web-scale probabilistic knowledge base using a variety of information extraction sources. Like other knowledge bases, KV stores information in triple form, (subject, predicate, object, confidence) for example the following triple represents the statement "Barak Obama was born in Honolulu":

```
</m/02mjmr,
  /people/person/place_of_birth /m/02hrh0_ 0.98>
```

where /m/02mjmr is the ID for Barak Obama and /m/02hrh0_ is the ID for Honolulu, while 0.98 is a probability representing the system's confidence in the triple[1].

The Knowledge Vault combines noisy extractions from the Web with prior knowledge derived from other knowledge bases. The size of KV was reported at 1.6 billion triples by Dong et al. [2014], and has since almost doubled in size. There are about 50M entities, and a few thousands relation types. The data is proprietary and to our knowledge has not been made publicly available.

Yago Yago is a project developed at MPI that consists of a large ontology automatically extracted from the Web, see e.g. the latest survey by Hoffart et al. [2013]. The ontology is automatically derived from Wikipedia WordNet and GeoNames, and currently has 10 million entities and more than 120 million facts (triples). While the publicly available data is determinstic, it is reported to us by Martin Theobald that intermediate stages of the system has data annotated with confidence values, expressed as probabilities.

Reverb Reverb is reported in Fader et al. [2011] and contains 15 million binary assertions from the Web. The data is available for download from the ReVerb homepage at http://reverb.cs.washington.edu/, and is now included in OpenIE http://openie.allenai.org/. The available dataset in ReVerb contains only extractions with a confidence value > 0.9.

6.2 Probabilistic Database Systems

Several probabilistic database systems have been reported in the literature. All systems faced the fundamental challenge of reconciling the inherent computational complexity of probabilistic inference, with the expectation on any database system to scale up to very large data sets; perhaps for that reason, no commercial, general purpose probabilistic database system exists to date.

[1]The actual confidence of this triple is not reported in Dong et al. [2014].

Trio The Trio system described by Benjelloun et al. [2006a] manages incomplete and probabilistic databases based on maybe-tuples, X-tuples, and lineage expressions. For query evaluation the system computes the query's lineage then uses an internally developed DPLL-based weighted model counter to compute the probability.

MayBMS MayBMS is a probabilistic database system described by Antova et al. [2007] and is available for download at `http://maybms.sourceforge.net/`. MayBMS is a modified postgres system that supports tuple-independent probabilistic relations with a confidence attribute and computes output probabilities for SQL queries. The probabilistic inference is done either using safe plans (when possible) or by performing weighted model counting using a DPLL-based algorithm.

MystiQ MystiQ is a probabilistic database system described by Ré and Suciu [2008], which supports tuple independent and BID relations. MystiQ is a lightweight interface connecting to postgres. It accepts SQL queries and either converts them to safe plans (expressed in SQL and run in postgres) or computes the lineage then computes (outside the engine) an approximate probability using Karp and Luby's FPTRAS.

Alchemy The first implementation of Markov Logic Network described by Richardson and Domingos [2006] is Alchemy. Users define soft and hard constraints, and provide *evidence*, which are ground tuples with probability 1. The system computes either the MAP or marginal probabilities and uses an MCMC algorithm to perform probabilistic inference. Alchemy is implemented entirely from scratch and does not use any database technology.

Tuffy and DeepDive An implementation of Markov Logic Networks that uses a database engine (postgres) to perform the grounding is described by Niu et al. [2011] and is called Tuffy. By using a standard query engine to perform grounding the authors report speedups

of orders of magnitude over Alchemy. Tuffy is is available for download at `http://i.stanford.edu/hazy/tuffy/download/`. It is now part of the larger DeepDive system [Zhang, 2015].

ProbLog ProbLog is a well-maintained probabilistic logic programming system. It supports datalog queries on probabilistic databases. A recent overview is described in Fierens et al. [2015]. The probabilities can also be associated to the rules rather than to the data, and ProbLog allows recursive queries. For example, the following ProbLog rule fires only with probability 0.3:

```
0.3::smoker(Y) :- smoker(X), friend(X,Y)
```

Query evaluation in ProbLog is based on SDDs. First, the system computes the lineage of the query, then the resulting formula is converted into an SDD; the probability is then computed on the SDD using standard dynamic programming (see §5.1). ProbLog is available from `https://dtai.cs.kuleuven.be/problog/`.

SlimShot SlimShot was introduced by Gribkoff and Suciu [2016] as a probabilistic database system that combines lifted inference with approximate weighted model counting based on Monte Carlo simulations. The system takes as input a set of soft constraints (Markov logic networks) and a query, and evaluates the query by using the reduction described in §2.6. Then it chooses a set of relation names such that both numerator and denominator of the conditional probability (Equation 2.11) are liftable, and performs collapsed particle sampling on only these relations. The system is available from `https://github.com/ericgribkoff/slimshot`.

ProbKB, ProbKB, described by Chen and Wang [2014], is a probabilistic database system for evaluating massive numbers of soft constraints. The system focuses on grounding the soft constraints, by exploiting the fact that in a large Knowledge Base the number of distinct rule templates is limited. For example, the 30,000 soft constraints automatically extracted by the ReVerb project conform to only 7 distinct templates, which ProbKb can ground by issuing only 7 SQL queries.

Forclift For lifted inference in *symmetric* probabilistic databases, Markov logic networks, and applications to lifted machine learning, the Forclift system implements algorithms for first-order knowledge compilation [Van den Broeck, 2013]. The system is available from `https://github.com/UCLA-StarAI/Forclift`.

6.3 Applications

Many applications include some form of probabilistic data, but only a few require query evaluation in the sense described in this survey. We briefly list here some of the latter.

Knowledge Base Construction (KBC) is the process of populating a structured relational database from unstructured sources: the system reads a large number of documents (Web pages, journal articles, news stories) and populates a relational database with facts. Shin et al. [2015] describe DeepDive, a system that uses a declarative mapping in a language that combines SQL with probabilities, using a semantics similar to Markov Logic Networks. DeepDive performs two major tasks. First, *grounding*, evaluates a large number of SQL queries to produce a large database called a factor graph. Second, *inference*, runs a large MCMC simulation on the factor graph. DeepDive is reported to take hours on a 1TB RAM/48-core machine to perform the probabilistic inference based on MCMC.

Inferring Missing Facts Several research projects have been developed to learn rules that can be used to infer more knowledge. Fader et al. [2011] describe SHERLOCK, a system that infers over 30,000 Horn clauses automatically from ground facts previously derived from Web text, using open information extraction. Learning is done in an un-supervised, domain-independent manner, based on techniques developed in the Inductive Logic Programming (ILP) literature. All 30,000 rules are *soft rules*, in the sense that they carry a weight representing the systems confidence in that rule. Coping with such a large number of rules is a major challenge. For example, just

grounding 30,000 rules requires running 30,000 SQL queries, which is prohibitively expensive; as we mentioned earlier, ProbKB, a project described by Chen and Wang [2014], speeds up grounding by classifying the rules into a small number of templates.

Bootstrapping Approximate Query Processing (AQP) consists of a suite of techniques to allow interactive data exploration over massive datasets. The main goal is to trade off query evaluation speed for precision: users are willing to settle for an approximate answer, if that answer can be computed at interactive speed. Some well know AQP systems offering this tradeoff are described by Hellerstein et al. [1997], Jermaine et al. [2007], Kandula et al. [2016], Agarwal et al. [2013], Ding et al. [2016]. However, the approximate answers need to be quantified with confidence intervals, and computing these intervals remains a major challenge in AQP. Bootstrapping is a classic technique in statistics to compute the confidence intervals, and consists of repeatedly re-sampling from the sample, with replacement, and observing the confidence interval of the query answer on the re-samples. However, the number of re-samples required is rather large, which in turn defeats the purpose of AQP of returning answers very quickly. Zeng et al. [2014] propose an alternative solution to bootstrapping, based on lifted probabilistic inference. Every tuple t in the sample is associated with a numerical random variable $X_t \in \{0, 1, 2, \ldots\}$ indicating how many re-samples selected that tuple. Then, if the query to be computed is liftable, its distribution can be obtained using adaptations of the lifted inference techniques described in Chapter 4.

Lifted Learning The weights of the soft constraints in a Markov logic network are typically learned from training data; moreover, some applications even require the soft rules themselves to be learned from data, and this is especially challenging given the huge space of possible theories. The learning task is an iterative process to compute the maximum likelihood of a model given the data, and the critical piece of the loop is probabilistic inference, i.e. computing the probability of the existing data given the current model. Van Haaren et al.

[2016] describe a system that uses lifted inference for the inner loop of the learning task. The authors report that the lifted learning algorithm results in more accurate models than several competing approximate approaches. In related work, Jaimovich et al. [2007] and Ahmadi et al. [2012] speed up Markov logic network parameter learning by performing approximate lifted message passing inference.

Image Retrieval Zhu et al. [2015] use probabilistic databases for image retrieval within the DeepDive system: given a textual query, such as "Find photos of me sea kayaking last Halloween in my photo album", the task is to show images that fit the description. Probabilistic databases are particularly suited to answer queries that require joint reasoning about several (probabilistic) image features and meta-data. Zhu et al. [2014] learn a more specific MLN knowledge base from images, in order to reason about object affordances.

7

Conclusions and Open Problems

This survey discussed two recent developments in probabilistic inference: query evaluation in probabilistic databases, and lifted inference in statistical relational models. In probabilistic databases the input is large, defines a simple probabilistic space, but the query generates a complex model, where probabilistic inference may be challenging. In statistical relational models, the probabilistic model is usually a large graphical model, such as a Markov network or a Bayesian network, which is specified using a much shorter first-order relational representation. We have explained why these two models are, essentially, equivalent, and have discussed the complexity of the probabilistic inference problem under various assumptions.

Probabilistic inference remains a major challenge in Computer Science, and is of increasing importance because many algorithms that process large amounts of data use probabilities to model the uncertainty in that data. For example, probabilistic inference remains the key technical bottleneck in the automated construction of large knowledge bases. However, large probabilistic models are almost always generated programatically, from some high level specification, and the goal of lifted inference is to speedup probabilistic inference by "lift-

ing" it to that high-level specification. We have seen in this survey recent advances, as well as limitations of lifted inference.

Probabilistic data processing and lifted inference offer a rich set of open problems, for any Databases or AI researcher interested in probabilistic inference, or any PhD student looking for a thesis topic. We mention here just a handful of open problems.

On the theoretical side, one is to study the complexity of approximate lifted inference. We have seen that there exists a query whose probability is NP-hard to approximate, and we have seen that every existentially quantified query admits an FPTRAS. The problem is: given any query, prove that it either has an FPTRAS or is NP-hard to approximate. On the other hand, some system do not compute marginal probabilities, but instead compute the MPD (Most Probable Database). The user specifies a set of (possibly soft) constraints in some high level language, and the task is to find the most likely world that satisfies the constraint. Since computing the MPD is known to be NP-hard in general, the question is to characterize the constraints for which the MPD can be computed efficiently. Ideally, in either case, one wishes to have a dichotomy theorem, but perhaps the dichotomy could be restricted to a small class of queries. Finally, we mention that the use of symmetries is still very poorly understood. We lack a general characterization of inference complexity over symmetric probabilistic databases, and even the data complexity class for such problems, $\#\mathrm{P}_1$, has not been the subject of sufficient study.

On the practical side, integrating the insights described here into a scalable general-purpose system is an open problem. Such a system should provide the user with enough modeling power and query capabilities to enable applications in information extraction, data cleaning, relational learning, network mining, etc. At the same time, it should retain the ability to perform lifted inference and find effective approximations, even for theoretically hard problems.

Acknowledgments

Guy Van den Broeck was partially supported by NSF grants IIS-1657613, IIS-1633857 and DARPA XAI grant N66001-17-2-4032. Dan Suciu was supported by NSF grant III-1614738.

References

Daniel Abadi, Peter A. Boncz, Stavros Harizopoulos, Stratos Idreos, and Samuel Madden. The design and implementation of modern column-oriented database systems. *Foundations and Trends in Databases*, 5(3):197–280, 2013.

Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

Sameer Agarwal et al. Blinkdb: queries with bounded errors and bounded response times on very large data. In *Proc. of EuroSys'13*, pages 29–42, 2013.

Babak Ahmadi, Kristian Kersting, and Sriraam Natarajan. Lifted online training of relational models with stochastic gradient methods. In *ECML PKDD*, pages 585–600, 2012.

Sheldon B. Akers Jr. Binary decision diagrams. *IEEE Trans. Computers*, 27(6): 509–516, 1978.

Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Provenance circuits for trees and treelike instances. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, pages 56–68, 2015.

Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Tractable lineages on treelike instances: Limits and extensions. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 355–370, 2016.

Ankit Anand, Ritesh Noothigattu, Parag Singla, and Mausam. Non-count symmetries in boolean & multi-valued prob. graphical models. In *Artificial Intelligence and Statistics*, pages 1541–1549, 2017.

Lyublena Antova, Christoph Koch, and Dan Olteanu. Maybms: Managing incomplete information with probabilistic world-set decompositions. In *ICDE*, pages 1479–1480, 2007.

F. Bacchus. Representing and reasoning with probabilistic knowledge: a logical approach to probabilities. 1991.

Vince Barany, Balder ten Cate, Benny Kimelfeld, Dan Olteanu, and Zografoula Vagena. Declarative probabilistic programming with datalog. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 48. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

Daniel Barbará, Hector Garcia-Molina, and Daryl Porter. The management of probabilistic data. *IEEE Trans. Knowl. Data Eng.*, 4(5):487–502, 1992.

Paul Beame and Vincent Liew. New limits for knowledge compilation and applications to exact model counting. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence, UAI 2015, July 12-16, 2015, Amsterdam, The Netherlands*, pages 131–140, 2015.

Paul Beame, Jerry Li, Sudeepa Roy, and Dan Suciu. Lower bounds for exact model counting and applications in probabilistic databases. In *UAI*, pages 157–162, 2013.

Paul Beame, Jerry Li, Sudeepa Roy, and Dan Suciu. Counting of query expressions: Limitations of propositional methods. In *ICDT*, pages 177–188, 2014.

Paul Beame, Jerry Li, Sudeepa Roy, and Dan Suciu. Exact model counting of query expressions: Limitations of propositional methods. *ACM TODS*, 2017. (to appear).

Omar Benjelloun, Anish Das Sarma, Alon Y. Halevy, and Jennifer Widom. ULDBs: Databases with uncertainty and lineage. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, pages 953–964, 2006a.

Omar Benjelloun, Anish Das Sarma, Chris Hayworth, and Jennifer Widom. An introduction to ULDBs and the Trio system. *IEEE Data Eng. Bull.*, 29 (1):5–16, 2006b.

Simone Bova. Sdds are exponentially more succinct than obdds. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 929–935, 2016.

Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

Hung Hai Bui, Tuyen N Huynh, Artificial Intelligence Center, and Sebastian Riedel. Automorphism groups of graphical models and lifted variational inference. In *UAI*, page 132, 2013.

Wray L. Buntine. Operations for learning with graphical models. *JAIR*, 2: 159–225, 1994.

Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr., and Tom M. Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, 2010.

Roger Cavallo and Michael Pittarelli. The theory of probabilistic databases. In *VLDB'87, Proceedings of 13th International Conference on Very Large Data Bases, September 1-4, 1987, Brighton, England*, pages 71–81, 1987.

Ismail Ilkan Ceylan, Adnan Darwiche, and Guy Van den Broeck. Open-world probabilistic databases. In *Proceedings of the 15th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 2016.

Supratik Chakraborty, Daniel J. Fremont, Kuldeep S. Meel, Sanjit A. Seshia, and Moshe Y. Vardi. Distribution-aware sampling and weighted model counting for SAT. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1722–1730, 2014.

Supratik Chakraborty, Dror Fried, Kuldeep S. Meel, and Moshe Y. Vardi. From weighted to unweighted model counting. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 689–695, 2015.

Surajit Chaudhuri. An overview of query optimization in relational systems. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington, USA*, pages 34–43, 1998.

M. Chavira and A. Darwiche. Compiling Bayesian networks with local structure. In *Proceedings of IJCAI*, volume 5, pages 1306–1312, 2005.

M. Chavira and A. Darwiche. On probabilistic inference by weighted model counting. *AIJ*, 172(6–7):772–799, 2008.

Yang Chen and Daisy Zhe Wang. Knowledge expansion over probabilistic knowledge bases. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 649–660, 2014.

Arthur Choi and Adnan Darwiche. An edge deletion semantics for belief propagation and its practical impact on approximation quality. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 1107. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

Arthur Choi and Adnan Darwiche. Relax, compensate and then recover. In *JSAI International Symposium on Artificial Intelligence*, pages 167–180. Springer, 2010.

Arthur Choi, Doga Kisa, and Adnan Darwiche. Compiling probabilistic graphical models using sentential decision diagrams. In *ECSQARU*, pages 121–132, 2013.

Jaesik Choi, Rodrigo de Salvo Braz, and Hung H. Bui. Efficient methods for lifted inference with aggregate factors. In *AAAI*, 2011.

Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.

Fabio G. Cozman. Credal networks. *AIJ*, 120(2):199–233, 2000. .

Fabio Gagliardi Cozman and Denis Deratani Mauá. Bayesian networks specified using propositional and relational constructs: Combined, data, and domain complexity. In *AAAI*, pages 3519–3525, 2015.

Fabio Gagliardi Cozman and Denis Deratani Mauá. Probabilistic graphical models specified by probabilistic logic programs: Semantics and complexity. In *Conference on Probabilistic Graphical Models*, pages 110–122, 2016.

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

Nilesh N. Dalvi and Dan Suciu. Management of probabilistic data: foundations and challenges. In *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11-13, 2007, Beijing, China*, pages 1–12, 2007a.

Nilesh N. Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. *VLDB J.*, 16(4):523–544, 2007b.

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

Adnan Darwiche. Any-space probabilistic inference. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 133–142. Morgan Kaufmann Publishers Inc., 2000.

Adnan Darwiche. Decomposable negation normal form. *J. ACM*, 48(4):608–647, 2001.

Adnan Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009.

Adnan Darwiche. SDD: A new canonical representation of propositional knowledge bases. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 819–826, 2011.

Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *J. Artif. Int. Res.*, 17(1):229–264, September 2002.

Adnan Darwiche, Rina Dechter, Arthur Choi, Vibhav Gogate, and Lars Otten. Results from the probablistic inference evaluation of UAI-08. 2008.

Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960.

Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

Luc De Raedt and Angelika Kimmig. Probabilistic (logic) programming concepts. *Machine Learning*, 100(1):5–47, 2015.

Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *Proceedings of the 20th international joint conference on Artifical intelligence*, pages 2468–2473, 2007.

Luc De Raedt, Kristian Kersting, Sriraam Natarajan, and David Poole. Statistical relational artificial intelligence: Logic, probability, and computation. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 10(2):1–189, 2016.

Rodrigo de Salvo Braz, Eyal Amir, and Dan Roth. Lifted first-order probabilistic inference. In *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, pages 1319–1325, 2005.

Rina Dechter and Irina Rish. Mini-buckets: A general scheme for bounded inference. *Journal of the ACM (JACM)*, 50(2):107–153, 2003.

Amol Deshpande, Carlos Guestrin, Samuel Madden, Joseph M. Hellerstein, and Wei Hong. Model-driven data acquisition in sensor networks. In *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004*, pages 588–599, 2004.

Bolin Ding, Silu Huang, Surajit Chaudhuri, Kaushik Chakrabarti, and Chi Wang. Sample+seek: Approximating aggregates with distribution precision guarantee. In *Proc. SIGMOD*, pages 679–694, 2016.

Pedro Domingos and Daniel Lowd. *Markov Logic: An Interface Layer for Artificial Intelligence*. Morgan & Claypool Publishers, 2009.

Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: a web-scale approach to probabilistic knowledge fusion. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 601–610, 2014.

Stefano Ermon, Carla P Gomes, Ashish Sabharwal, and Bart Selman. Optimization with parity constraints: from binary codes to discrete integration. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*, pages 202–211. AUAI Press, 2013.

Oren Etzioni, Michele Banko, Stephen Soderland, and Daniel S. Weld. Open information extraction from the web. *Commun. ACM*, 51(12):68–74, 2008.

Anthony Fader, Stephen Soderland, and Oren Etzioni. Identifying relations for open information extraction. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, EMNLP 2011, 27-31 July 2011, John McIntyre Conference Centre, Edinburgh, UK, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1535–1545, 2011.

Ronald Fagin, Joseph Y. Halpern, and Nimrod Megiddo. A logic for reasoning about probabilities. *Inf. Comput.*, 87(1/2):78–128, 1990.

Daan Fierens, Hendrik Blockeel, Maurice Bruynooghe, and Jan Ramon. Logical Bayesian networks and their relation to other probabilistic logical models. *Inductive Logic Programming*, pages 121–135, 2005.

Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Sht. Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. Inference and learning in probabilistic logic programs using weighted boolean formulas. *TPLP*, 15(3):358–401, 2015. .

Robert Fink and Dan Olteanu. A dichotomy for non-repeating queries with negation in probabilistic databases. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '14, pages 144–155, New York, NY, USA, 2014. ACM. .

Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*, pages 1300–1309, 1999.

Norbert Fuhr. A probabilistic relational model for the integration of IR and databases. In *Proceedings of the 16th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval. Pittsburgh, PA, USA, June 27 - July 1, 1993*, pages 309–317, 1993. .

Norbert Fuhr and Thomas Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.*, 15(1):32–66, 1997.

Wolfgang Gatterbauer and Dan Suciu. Oblivious bounds on the probability of boolean functions. *ACM Trans. Database Syst.*, 39(1):5, 2014.

Wolfgang Gatterbauer and Dan Suciu. Approximate lifted inference with probabilistic databases. *PVLDB*, 8(5):629–640, 2015.

Erol Gelenbe and Georges Hébrail. A probability model of uncertainty in data bases. In *Proceedings of the Second International Conference on Data Engineering, February 5-7, 1986, Los Angeles, California, USA*, pages 328–333, 1986.

Lise Getoor and Ben Taskar. *Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning series)*. MIT Press, 2007.

Lise Getoor, Benjamin Taskar, and Daphne Koller. Selectivity estimation using probabilistic models. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data, Santa Barbara, CA, USA, May 21-24, 2001*, pages 461–472, 2001.

Boris Glavic and Gustavo Alonso. Perm: Processing provenance and data on the same data model through query rewriting. In *Proceedings of the 25th International Conference on Data Engineering, ICDE*, pages 174–185, 2009.

Vibhav Gogate and Pedro Domingos. Probabilistic theorem proving. In *UAI*, pages 256–265, 2011.

Martin Charles Golumbic, Aviad Mintz, and Udi Rotics. Factoring and recognition of read-once functions using cographs and normality and the readability of functions associated with partial k-trees. *Discrete Applied Mathematics*, 154(10):1465–1477, 2006.

Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Model counting. In *Handbook of Satisfiability*, pages 633–654. 2009.

Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. Church: A language for generative models. In *Proceedings of UAI*, pages 220–229, 2008.

Goetz Graefe. Query evaluation techniques for large databases. *ACM Comput. Surv.*, 25(2):73–170, 1993.

Todd J. Green and Val Tannen. Models for incomplete and probabilistic information. *IEEE Data Eng. Bull.*, 29(1):17–24, 2006.

Eric Gribkoff and Dan Suciu. Slimshot: In-database probabilistic inference for knowledge bases. *PVLDB*, 9(7):552–563, 2016.

Eric Gribkoff, Guy Van den Broeck, and Dan Suciu. Understanding the complexity of lifted inference and asymmetric weighted model counting. In *UAI*, pages 280–289, 2014a.

Eric Gribkoff, Dan Suciu, and Guy Van den Broeck. Lifted probabilistic inference: A guide for the database researcher. *IEEE Data Eng. Bull.*, 37(3): 6–17, 2014b.

Eric Gribkoff, Guy Van den Broeck, and Dan Suciu. The most probable database problem. In *Proceedings of the First International Workshop on Big Uncertain Data (BUDA)*, June 2014c.

Martin Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*. LNCS, 2017. (to appear).

Joseph Y. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, 46(3):311–350, 1990.

Joseph Y. Halpern. *Reasoning about uncertainty*. MIT Press, 2003.

Jochen Heinsohn. Probabilistic description logics. In *Proceedings of the Tenth international conference on Uncertainty in artificial intelligence*, pages 311–318. Morgan Kaufmann Publishers Inc., 1994.

Joseph M. Hellerstein, Peter J. Haas, and Helen J. Wang. Online aggregation. In *Proc. of SIGMOD*, pages 171–182, 1997.

W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artif. Intell.*, 194:28–61, 2013.

Michael C. Horsch and David L. Poole. A dynamic approach to probabilistic inference. In *Proceedings of UAI*, 1990.

Jinbo Huang and Adnan Darwiche. Dpll with a trace: From sat to knowledge compilation. In *IJCAI*, pages 156–162, 2005.

Edward Hung, Lise Getoor, and VS Subrahmanian. Probabilistic interval xml. In *International Conference on Database Theory*, pages 361–377. Springer, 2003.

Tomasz Imielinski and Witold Lipski, Jr. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984.

Russell Impagliazzo and Valentine Kabanets. Constructive proofs of concentration bounds. In Maria J. Serna, Ronen Shaltiel, Klaus Jansen, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 13th International Workshop, APPROX 2010, and 14th International Workshop, RANDOM 2010, Barcelona, Spain, September 1-3, 2010. Proceedings*, volume 6302 of *Lecture Notes in Computer Science*, pages 617–631. Springer, 2010.

Manfred Jaeger. Relational Bayesian networks. In *Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence*, pages 266–273. Morgan Kaufmann Publishers Inc., 1997.

Manfred Jaeger. On the complexity of inference about probabilistic relational models. *Artificial Intelligence*, 117(2):297–308, 2000.

Manfred Jaeger and Guy Van den Broeck. Liftability of probabilistic inference: Upper and lower bounds. In *Proceedings of the 2nd International Workshop on Statistical Relational AI*, 2012.

Ariel Jaimovich, Ofer Meshi, and Nir Friedman. Template based inference in symmetric relational markov random fields. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, pages 191–199. AUAI Press, 2007.

Ravi Jampani, Fei Xu, Mingxi Wu, Luis Leopoldo Perez, Christopher Jermaine, and Peter J Haas. Mcdb: a monte carlo approach to managing uncertain data. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 687–700. ACM, 2008.

Christopher M. Jermaine, Subramanian Arumugam, Abhijit Pol, and Alin Dobra. Scalable approximate query processing with the DBO engine. In *Proc. of SIGMOD*, pages 725–736, 2007.

Yacine Jernite, Alexander M Rush, and David Sontag. A fast variational approach for learning Markov random field language models. In *ICML*, 2015.

Abhay Jha, Vibhav Gogate, Alexandra Meliou, and Dan Suciu. Lifted inference seen from the other side: The tractable features. In *NIPS*, pages 973–981, 2010.

Abhay Kumar Jha and Dan Suciu. Knowledge compilation meets database theory: compiling queries to decision diagrams. In *Database Theory - ICDT 2011, 14th International Conference, Uppsala, Sweden, March 21-24, 2011, Proceedings*, pages 162–173, 2011.

Abhay Kumar Jha and Dan Suciu. Probabilistic databases with markoviews. *PVLDB*, 5(11):1160–1171, 2012.

Abhay Kumar Jha and Dan Suciu. Knowledge compilation meets database theory: Compiling queries to decision diagrams. *Theory Comput. Syst.*, 52 (3):403–440, 2013.

Bhargav Kanagal, Jian Li, and Amol Deshpande. Sensitivity analysis and explanations for robust query evaluation in probabilistic databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*, pages 841–852, 2011.

Srikanth Kandula, Anil Shanbhag, Aleksandar Vitorovic, Matthaios Olma, Robert Grandl, Surajit Chaudhuri, and Bolin Ding. Quickr: Lazily approximating complex adhoc queries in bigdata clusters. In *Proc. of SIGMOD*, pages 631–646, 2016.

Richard M. Karp and Michael Luby. Monte-carlo algorithms for enumeration and reliability problems. In *24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November 1983*, pages 56–64, 1983.

Seyed Mehran Kazemi and David Poole. Elimination ordering in first-order probabilistic inference. In *AAAI*, 2014.

Seyed Mehran Kazemi and David Poole. Knowledge compilation for lifted probabilistic inference: Compiling to a low-level language. In *KR*, 2016.

Seyed Mehran Kazemi, Angelika Kimmig, Guy Van den Broeck, and David Poole. New liftable classes for first-order probabilistic inference. In *Advances in Neural Information Processing Systems 29 (NIPS)*, December 2016.

Gabriele Kern-Isberner and Thomas Lukasiewicz. Combining probabilistic logic programming with the power of maximum entropy. *Artificial Intelligence*, 157(1-2):139–202, 2004.

Kristian Kersting and Luc De Raedt. Bayesian logic programs. *arXiv preprint cs/0111058*, 2001.

Kristian Kersting, Babak Ahmadi, and Sriraam Natarajan. Counting belief propagation. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 277–284. AUAI Press, 2009.

Benny Kimelfeld and Yehoshua Sagiv. Matching twigs in probabilistic xml. In *Proceedings of the 33rd international conference on Very large data bases*, pages 27–38. VLDB Endowment, 2007.

Angelika Kimmig, Stephen Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. A short introduction to probabilistic soft logic. In *Proceedings of the NIPS Workshop on Probabilistic Programming: Foundations and Applications*, pages 1–4, 2012.

Timothy Kopp, Parag Singla, and Henry Kautz. Lifted symmetry detection and breaking for MAP inference. In *NIPS*, pages 1315–1323, 2015.

Donald Kossmann. The state of the art in distributed query processing. *ACM Comput. Surv.*, 32(4):422–469, 2000.

Laks V. S. Lakshmanan, Nicola Leone, Robert B. Ross, and V. S. Subrahmanian. Probview: A flexible probabilistic database system. *ACM Trans. Database Syst.*, 22(3):419–469, 1997.

C. Y. Lee. Representation of switching circuits by binary-decision programs. *Bell System Technical Journal*, 38:985–999, 1959.

Jian Li and Amol Deshpande. Consensus answers for queries over probabilistic databases. In *Proceedings of the Twenty-Eigth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2009, June 19 - July 1, 2009, Providence, Rhode Island, USA*, pages 259–268, 2009. .

Jian Li, Barna Saha, and Amol Deshpande. A unified approach to ranking in probabilistic databases. *VLDB J.*, 20(2):249–275, 2011.

Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004.

Leonid Libkin. SQL's three-valued logic and certain answers. In *18th International Conference on Database Theory, ICDT 2015, March 23-27, 2015, Brussels, Belgium*, pages 94–109, 2015.

Daniel Lowd and Pedro Domingos. Efficient weight learning for markov logic networks. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 200–211. Springer, 2007.

Thomas Lukasiewicz. Expressive probabilistic description logics. *Artificial Intelligence*, 172(6):852–883, 2008.

Andrew McCallum, Karl Schultz, and Sameer Singh. Factorie: Probabilistic programming via imperatively defined factor graphs. In *Neural Information Processing Systems (NIPS)*, 2009.

Lilyana Mihalkova and Raymond J. Mooney. Bottom-up learning of markov logic network structure. In *Proceedings of the 24th international conference on Machine learning*, pages 625–632. ACM, 2007.

Gerome Miklau and Dan Suciu. A formal analysis of information disclosure in data exchange. *J. Comput. Syst. Sci.*, 73(3):507–534, 2007.

B. Milch, B. Marthi, S. Russell, D. Sontag, D.L. Ong, and A. Kolobov. BLOG: Probabilistic models with unknown objects. *Introduction to statistical relational learning*, page 373, 2007.

Brian Milch, Luke S. Zettlemoyer, Kristian Kersting, Michael Haimes, and Leslie Pack Kaelbling. Lifted probabilistic inference with counting formulae. In *AAAI*, pages 1062–1068, 2008.

Tom M. Mitchell, William W. Cohen, Estevam R. Hruschka Jr., Partha Pratim Talukdar, Justin Betteridge, Andrew Carlson, Bhavana Dalvi Mishra, Matthew Gardner, Bryan Kisiel, Jayant Krishnamurthy, Ni Lao, Kathryn Mazaitis, Thahir Mohamed, Ndapandula Nakashole, Emmanouil Antonios Platanios, Alan Ritter, Mehdi Samadi, Burr Settles, Richard C. Wang, Derry Tanti Wijaya, Abhinav Gupta, Xinlei Chen, Abulhair Saparov, Malcolm Greaves, and Joel Welling. Never-ending learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 2302–2310, 2015.

Katherine F. Moore, Vibhor Rastogi, Christopher Ré, and Dan Suciu. Query containment of tier-2 queries over a probabilistic database. In *Proceedings of the Third VLDB workshop on Management of Uncertain Data (MUD2009) in conjunction with VLDB 2009, Lyon, France, August 28th, 2009.*, pages 47–62, 2009.

S. Muggleton. Stochastic logic programs. *Advances in inductive logic programming*, 32:254–264, 1996.

Hung Q. Ngo, Christopher Ré, and Atri Rudra. Skew strikes back: new developments in the theory of join algorithms. *SIGMOD Record*, 42(4):5–16, 2013.

Liem Ngo and Peter Haddawy. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 171(1):147–177, 1997.

Mathias Niepert. Markov chains on orbits of permutation groups. In *UAI*, 2012.

Mathias Niepert and Guy Van den Broeck. Tractability through exchangeability: A new perspective on efficient probabilistic inference. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence, AAAI Conference on Artificial Intelligence*, July 2014.

Andrew Nierman and HV Jagadish. Protdb: Probabilistic data in xml. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 646–657. VLDB Endowment, 2002.

Nils J. Nilsson. Probabilistic logic. *Artificial intelligence*, 28(1):71–87, 1986.

Davide Nitti, Tinne De Laet, and Luc De Raedt. Probabilistic logic programming for hybrid relational domains. *Machine Learning*, 103(3):407–449, 2016. ISSN 1573-0565. .

Feng Niu, Christopher Ré, AnHai Doan, and Jude W. Shavlik. Tuffy: Scaling up statistical inference in markov logic networks using an rdbms. *PVLDB*, 4(6):373–384, 2011.

Dan Olteanu and Jiewen Huang. Using obdds for efficient query evaluation on probabilistic databases. In *Scalable Uncertainty Management, Second International Conference, SUM 2008, Naples, Italy, October 1-3, 2008. Proceedings*, pages 326–340, 2008.

Dan Olteanu and Jiewen Huang. Secondary-storage confidence computation for conjunctive queries with inequalities. In *SIGMOD Conference*, pages 389–402, 2009.

Dan Olteanu, Jiewen Huang, and Christoph Koch. Sprout: Lazy vs. eager query plans for tuple-independent probabilistic databases. In *ICDE*, pages 640–651, 2009.

Dan Olteanu, Jiewen Huang, and Christoph Koch. Approximate confidence computation in probabilistic databases. In *Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, March 1-6, 2010, Long Beach, California, USA*, pages 145–156, 2010.

M.A. Paskin. Maximum entropy probabilistic logic. Technical Report UCB/CSD-01-1161, Computer Science Division, University of California, Berkeley, 2002.

Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann, 1988.

Frederick E Petry. *Fuzzy databases: principles and applications*, volume 5. Springer Science & Business Media, 2012.

Avi Pfeffer. Figaro: An object-oriented probabilistic programming language. *Charles River Analytics Technical Report*, 2009.

David Poole. Probabilistic horn abduction and Bayesian networks. *Artificial intelligence*, 64(1):81–129, 1993.

David Poole. The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94(1):7–56, 1997.

David Poole. First-order probabilistic inference. In *IJCAI*, volume 3, pages 985–991, 2003.

J. Scott Provan and Michael O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput.*, 12 (4):777–788, 1983.

Christopher Ré and Dan Suciu. Materialized views in probabilistic databases for information exchange and query optimization. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, pages 51–62, 2007.

Christopher Ré and Dan Suciu. Managing probabilistic data with mystiq: The can-do, the could-do, and the can't-do. In *Scalable Uncertainty Management, Second International Conference, SUM 2008, Naples, Italy, October 1-3, 2008. Proceedings*, pages 5–18, 2008.

Christopher Ré and Dan Suciu. The trichotomy of HAVING queries on a probabilistic database. *VLDB J.*, 18(5):1091–1116, 2009.

Christopher Ré, Nilesh N. Dalvi, and Dan Suciu. Efficient top-k query evaluation on probabilistic data. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, pages 886–895, 2007.

Raymond Reiter. On closed world data bases. *Logic and Data Bases*, pages 55–76, 1978.

Joris Renkens, Guy Van den Broeck, and Siegfried Nijssen. k-optimal: a novel approximate inference algorithm for problog. *Machine learning*, 89(3):215–231, 2012.

Joris Renkens, Angelika Kimmig, Guy Van den Broeck, and Luc De Raedt. Explanation-based approximate weighted model counting for probabilistic logics. In *AAAI Workshop: Statistical Relational Artificial Intelligence*, 2014.

Matthew Richardson and Pedro M. Domingos. Markov logic networks. *Machine Learning*, 62(1-2):107–136, 2006.

Fabrizio Riguzzi. A top down interpreter for lpad and cp-logic. In *Congress of the Italian Association for Artificial Intelligence*, pages 109–120. Springer, 2007.

Dan Roth. On the hardness of approximate reasoning. *Artif. Intell.*, 82(1-2): 273–302, 1996.

Stuart J. Russell. Unifying logic and probability. *Commun. ACM*, 58(7):88–97, 2015.

Tian Sang, Fahiem Bacchus, Paul Beame, Henry A. Kautz, and Toniann Pitassi. Combining component caching and clause learning for effective model counting. In *SAT*, 2004.

Taisuke Sato. A statistical learning method for logic programs with distribution semantics. In *Proceedings of the 12th International Conference on Logic Programming (ICLP)*, pages 715–729. MIT Press, 1995.

Taisuke Sato and Yoshitaka Kameya. PRISM: a language for symbolic-statistical modeling. In *Proceedings of the International Joint Conference on Artificial Intelligence*, volume 15, pages 1330–1339, 1997.

Stefan Schoenmackers, Jesse Davis, Oren Etzioni, and Daniel S. Weld. Learning first-order horn clauses from web text. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, EMNLP 2010, 9-11 October 2010, MIT Stata Center, Massachusetts, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1088–1098, 2010.

Prithviraj Sen, Amol Deshpande, and Lise Getoor. Bisimulation-based approximate lifted inference. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 496–505. AUAI Press, 2009.

Pierre Senellart and Serge Abiteboul. On the complexity of managing probabilistic xml data. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 283–292. ACM, 2007.

Jaeho Shin, Sen Wu, Feiran Wang, Christopher De Sa, Ce Zhang, and Christopher Ré. Incremental knowledge base construction using deepdive. *PVLDB*, 8(11):1310–1321, 2015.

Sarvjeet Singh, Chris Mayfield, Sagar Mittal, Sunil Prabhakar, Susanne E. Hambrusch, and Rahul Shah. Orion 2.0: native support for uncertain data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 1239–1242, 2008.

Parag Singla and Pedro M Domingos. Lifted first-order belief propagation. In *AAAI*, volume 8, pages 1094–1099, 2008.

Richard P. Stanley. *Enumerative Combinatorics*. Cambridge University Press, 1997.

Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

Nima Taghipour, Jesse Davis, and Hendrik Blockeel. First-order decomposition trees. In *Advances in Neural Information Processing Systems*, pages 1052–1060, 2013.

Octavian Udrea, V.S. Subrahmanian, and Zoran Majkic. Probabilistic RDF. In *2006 IEEE International Conference on Information Reuse & Integration*, pages 172–177. IEEE, 2006.

Salil P. Vadhan. The complexity of counting in sparse, regular, and planar graphs. *SIAM J. Comput.*, 31(2):398–427, 2001.

Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 1979a.

Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979b.

Guy Van den Broeck. On the completeness of first-order knowledge compilation for lifted probabilistic inference. In *Advances in Neural Information Processing Systems 24 (NIPS),*, pages 1386–1394, 2011.

Guy Van den Broeck. *Lifted Inference and Learning in Statistical Relational Models*. PhD thesis, KU Leuven, January 2013.

Guy Van den Broeck. Towards high-level probabilistic reasoning with lifted inference. In *Proceedings of the AAAI Spring Symposium on KRR*, 2015.

Guy Van den Broeck and Adnan Darwiche. On the complexity and approximation of binary evidence in lifted inference. In *Advances in Neural Information Processing Systems 26 (NIPS)*, December 2013.

Guy Van den Broeck and Jesse Davis. Conditioning in first-order knowledge compilation and lifted probabilistic inference. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, pages 1–7. AAAI Press, 2012.

Guy Van den Broeck and Mathias Niepert. Lifted probabilistic inference for asymmetric graphical models. In *Proceedings of the 29th Conference on Artificial Intelligence (AAAI)*, 2015.

Guy Van den Broeck, Nima Taghipour, Wannes Meert, Jesse Davis, and Luc De Raedt. Lifted probabilistic inference by first-order knowledge compilation. In *IJCAI*, pages 2178–2185, 2011.

Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. Lifted relax, compensate and then recover: From approximate to exact lifted probabilistic inference. In *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2012.

Guy Van den Broeck, Wannes Meert, and Adnan Darwiche. Skolemization for weighted first-order model counting. In *KR*, 2014.

Jan Van Haaren, Guy Van den Broeck, Wannes Meert, and Jesse Davis. Lifted generative learning of Markov logic networks. *Machine Learning*, 103(1): 27–55, 2016. .

J. Vennekens, S. Verbaeten, and M. Bruynooghe. Logic programs with annotated disjunctions. *Logic Programming*, pages 95–119, 2004.

J. Vennekens, M. Denecker, and M. Bruynooghe. CP-logic: A language of causal probabilistic events and its relation to logic programming. *Theory and Practice of Logic Programming*, 9(3):245–308, 2009.

Deepak Venugopal and Vibhav G. Gogate. Scaling-up importance sampling for Markov logic networks. In *Advances in Neural Information Processing Systems*, pages 2978–2986, 2014.

Jonas Vlasselaer, Guy Van den Broeck, Angelika Kimmig, Wannes Meert, and Luc De Raedt. Anytime inference in probabilistic logic programs with Tp-compilation. In *Proceedings of 24th International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

William Yang Wang, Kathryn Mazaitis, and William 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*, pages 2129–2138. ACM, 2013.

Ingo Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM, 2000.

Michael Wick, Andrew McCallum, and Gerome Miklau. Scalable probabilistic databases with factor graphs and mcmc. *Proceedings of the VLDB Endowment*, 3(1-2):794–804, 2010.

Wentao Wu, Hongsong Li, Haixun Wang, and Kenny Qili Zhu. Probase: a probabilistic taxonomy for text understanding. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 481–492, 2012.

Kai Zeng, Shi Gao, Barzan Mozafari, and Carlo Zaniolo. The analytical bootstrap: a new method for fast error estimation in approximate query processing. In *SIGMOD Conference*, pages 277–288, 2014.

Ce Zhang. *DeepDive: a data management system for automatic knowledge base construction*. PhD thesis, 2015.

Xi Zhang and Jan Chomicki. On the semantics and evaluation of top-k queries in probabilistic databases. In *Proceedings of the 24th International Conference on Data Engineering Workshops, ICDE 2008, April 7-12, 2008, Cancún, México*, pages 556–563, 2008.

Yuke Zhu, Alireza Fathi, and Li Fei-Fei. Reasoning about object affordances in a knowledge base representation. In *European conference on computer vision*, pages 408–424. Springer, 2014.

Yuke Zhu, Ce Zhang, Christopher Ré, and Li Fei-Fei. Building a large-scale multimodal knowledge base system for answering visual queries. *arXiv preprint arXiv:1507.05670*, 2015.