# Querying and Learning in Probabilistic Databases

Maximilian Dylla[1], Martin Theobald[2], and Iris Miliaraki[3]

[1] Max Planck Institute for Informatics, Saarbrücken, Germany
[2] University of Antwerp, Antwerp, Belgium
[3] Yahoo Labs, Barcelona, Spain

**Abstract.** Probabilistic Databases (PDBs) lie at the expressive intersection of databases, first-order logic, and probability theory. PDBs employ logical deduction rules to process Select-Project-Join (SPJ) queries, which form the basis for a variety of declarative query languages such as Datalog, Relational Algebra, and SQL. They employ logical consistency constraints to resolve data inconsistencies, and they represent query answers via logical lineage formulas (aka. "data provenance") to trace the dependencies between these answers and the input tuples that led to their derivation. While the literature on PDBs dates back to more than 25 years of research, only fairly recently the key role of lineage for establishing a closed and complete representation model of relational operations over this kind of probabilistic data was discovered. Although PDBs benefit from their efficient and scalable database infrastructures for data storage and indexing, they couple the data computation with probabilistic inference, the latter of which remains a #P-hard problem also in the context of PDBs.
In this chapter, we provide a review on the key concepts of PDBs with a particular focus on our own recent research results related to this field. We highlight a number of ongoing research challenges related to PDBs, and we keep referring to an information extraction (IE) scenario as a running application to manage uncertain and temporal facts obtained from IE techniques directly inside a PDB setting.

**Keywords:** Probabilistic and Temporal Databases, Deduction Rules, Consistency Constraints, Information Extraction

## 1 Introduction

Over the past decade, the demand for managing structured, relational data has continued to increase at an unprecedented rate, as we are crossing the "Big Data" era. Database architectures of all kinds play a key role for managing this explosion of data, thus aiming to provide efficient storage, querying, and update functionalities at scale. One of the main initial assumptions in databases, however, is that all data stored in the database is deterministic. That is, a data item (or "tuple") either holds as a real-world piece of truth or it is absent from

the database. In reality, a large amount of the data that is supposed to be captured in a database is inherently noisy or otherwise uncertain. Example applications that deal with uncertain data range from scientific data management and sensor networks to data integration and knowledge management systems. Any sensor, for example, can only provide a limited precision, and hence its measurements are inherently uncertain with respect to the precise physical value. Also, even the currently most sophisticated information extraction (IE) methods can extract facts with particular degree of confidence. This is partly due to the ambiguity of sentences formulated in natural language, but mainly due to the heuristic nature of many extractions tools which often rely on hand-crafted regular expressions and various other forms of rule- or learning-based extraction techniques [9,6,64,35].

As a result of the efforts for handling uncertain data directly inside a scalable database architecture, the field of probabilistic databases (PDBs) has evolved as an established area of database research in recent years [65]. PDBs lie in the intersection of database systems [2,34] (for handling large amounts of data), first-order logic [62,67] (for formulating expressive queries and constraints over the captured data items), and probability theory [27,60] (for quantifying the uncertainty and coupling the relational operations with different kinds of probabilistic inference). So far, most research efforts in the field of PDBs have focused on the representation of uncertain, relational data on the one hand, thus designing appropriate data models, and on efficiently answering queries over this kind of data on the other hand, thus proposing suitable methods for query evaluation. Regarding the data model, a variety of approaches for compactly representing data uncertainty have been presented. One of the most popular approaches, which forms also the basis for this chapter, is that of a *tuple-independent PDB* [15,65], in which a probability value is attached to each tuple in the database, and all tuples are assumed to be independent of each other. More expressive models, such as *pc-tables* [33], have been proposed as well, where each tuple is annotated by a logical formula that captures the tuple's dependencies to other tuples in the database. Finally, there are also more sophisticated models which capture *statistical correlations* among the database tuples [39,54,58].

**Temporal-Probabilistic Databases.** Besides potentially being uncertain, data items can also be annotated by other dimensions such as time or location. Such techniques are already partly supported by traditional database systems, where temporal databases (TDBs) [37] have been an active research field for many years. To enable this kind of temporal data and temporal reasoning also in a PDB context, the underlying probabilistic models need to be extended to support additional data dimensions. As part of this chapter, we thus also focus on the intersection of temporal and probabilistic databases, i.e., capturing data that is valid during a specific time interval with a given probability. In this context, we present a *unified temporal-probabilistic database* (TPDB) *model* [23] in which both time and probability are considered as first-class citizens.

**Top-$k$ Query Processing.** Query evaluation in PDBs involves—apart from the common data computation step, found also in deterministic databases—an

additional probability computation step for computing the marginal probabilities of the respective query answers. While the complexity for the data computation step for any given SQL query is polynomial in the size of the underlying database, even fairly simple Select-Project-Join (SPJ) queries can involve an exponential cost in the probability computation step. In fact, the query evaluation problem in PDBs is known to be $\#\mathcal{P}$-hard [16,32]. Thus, efficient strategies for probability computations and the early pruning of low-probability query answers remains a key challenge for the scalable management of probabilistic data. Recent works on efficient probability computation in PDBs have addressed this problem mostly from two ends. The first group of approaches have restricted the class of queries, i.e., by focusing on *safe query plans*[16,14,17], or by considering a specific class of tuple-dependencies, commonly referred to as *read-once functions* [59]. In particular the second group of approaches allows for applying *top-k style pruning methods* [51,50,8,25] at the time when the query is processed. This alternative way of addressing probability computations aims to efficiently identify the top-$k$ most probable query answers. To achieve this they rely on lower and upper bounds for the probabilities of these answers, to avoid an exact computation of their probabilities.

**Learning Tuple Probabilities.** While most works in PDBs assume that the initial probabilities are provided as input along with the data items, in reality, an update or estimation of the tuple's input probabilities often is highly desirable. To this end, enabling such a learning approach for tuple probabilities is an important building block for many applications, such as creating, updating, or cleaning a PDB. Although this has already been stated as a key challenge by Dalvi et al. [13], to date, only very few works [63,43] explicitly tackle the problem of creating or updating a PDB. Our recent work [24], which is also presented in the context of this chapter, thus can be seen as one of the first works that addresses the *learning of tuple probabilities* in a PDB setting.

In brief, this chapter aims to provide an overview of the key concepts of PDB systems, the main challenges that need to be addressed to efficiently manage large amounts of uncertain data, and the different methods that have been proposed for dealing with these challenges. In this context, we provide an overview of our own recent results [25,22,24] related to this field. As a motivating and running example, we continue to refer to a (simplified) IE scenario, where factual knowledge is extracted from both structured and semistructured Web sources, which is a process that inherently results in large amounts of uncertain (and temporal) facts.

### 1.1 Running Application: Information Extraction

As a guiding theme for this chapter, we argue that one of the main application domains of PDBs—and in fact a major challenge for scaling these techniques to very large relational data collections—is *information extraction* [70]. The goal of IE is to harvest factual knowledge from semistructured sources, and even from free-text, to turn this knowledge into a more machine-readable format—in other words, to "turn text into database tuples". For example, the sentence

"Spielberg won the Academy Award for Best Director for Schindler's List (1993) and Saving Private Ryan (1998)" from Steven Spielberg's Wikipedia article[4], entails the fact that *Spielberg* won an *AcademyAward*, which we could represent as *WonAward*(*Spielberg*, *AcademyAward*).

Due to the many ways of rephrasing such statements in natural language, an automatic machinery that mines such facts from textual sources will inherently produce a number of erroneous extractions. Thus, the resulting knowledge base is never going to be 100% clean but rather remains to some degree uncertain. Since the Web is literally full of text and facts, managing the vast amounts of extracted facts in a *scalable way* and at the same time providing *high-confidence query answers* from potentially noisy and uncertain input data will remain a major challenge of any knowledge management system, including PDBs.

For an illustration, we model a simple IE workflow in a PDB. Usually, candidates for facts in sentences are detected by textual patterns [9,44]. For instance, for winning an award, the verb "won" might indeed be a good indicator. In our PDB, we want to capture the different ingredients that lead to the extraction of a fact. Besides the textual pattern, this could also involve the Web domain (such as *Wikipedia.org*), where we found the sentence of interest. Hence, we store these in separate probabilistic relations as shown in Figure 1. Therefore,

WonPrizeExtraction

| | Subject | Object | Pid | Did | $p$ |
|---|---|---|---|---|---|
| $I_1$ | *Spielberg* | *AcademyAward* | 1 | 1 | 1.0 |
| $I_2$ | *Spielberg* | *AcademyAward* | 2 | 1 | 1.0 |

BornInExtraction

| | Subject | Object | Pid | Did | $p$ |
|---|---|---|---|---|---|
| $I_3$ | *Spielberg* | *Cincinnati* | 3 | 1 | 1.0 |
| $I_4$ | *Spielberg* | *LosAngeles* | 3 | 2 | 1.0 |

UsingPattern

| | Pid | Pattern | $p$ |
|---|---|---|---|
| $I_5$ | 1 | *Received* | 0.8 |
| $I_6$ | 2 | *Won* | 0.5 |
| $I_7$ | 3 | *Born* | 0.9 |

FromDomain

| | Did | Domain | $p$ |
|---|---|---|---|
| $I_8$ | 1 | *Wikipedia.org* | 0.9 |
| $I_9$ | 2 | *Imdb.com* | 0.8 |

**Fig. 1.** An Example Probabilistic Database for an Information Extraction Setting

the probabilities of the tuples of each domain and each pattern reflect our trust in this source and pattern, respectively. To reconcile the facts along with their resulting probabilities from the PDB of Figure 1, we employ two deduction rules. In essence, they formulate a natural join on the *Pid* and *Did* columns of the underlying relations to connect an extraction pattern and an extraction domain to

[4] `http://en.wikipedia.org/wiki/Steven_Spielberg` (as of December, 2013)

the actual fact:

$$WonPrize(S,O) \leftarrow \begin{pmatrix} WonPrizeExtraction(S,O,Pid,Did) \\ \wedge\ UsingPattern(Pid,P) \\ \wedge\ FromDomain(Did,D) \end{pmatrix} \quad (1)$$

$$BornIn(S,O) \leftarrow \begin{pmatrix} BornInExtraction(S,O,Pid,Did) \\ \wedge\ UsingPattern(Pid,P) \\ \wedge\ FromDomain(Did,D) \end{pmatrix} \quad (2)$$

If we execute a query on the resulting *WonPrize* or *BornIn* relations, then the probabilities of the pattern and domain establish the probability of each answer fact with respect to the relational operations that were involved to obtain these query answers.

### 1.2 Challenges & Outline

A number of PDB systems have been released as open-source prototypes recently. These include systems like *MayBMS* [5], *MystiQ* [8], *Orion* [61], *PrDB* [58], *SPROUT* [48], and *Trio* [7], which all allow for storing and querying uncertain, relational data and meanwhile found a wide recognition in the database community. However, in order to make PDB systems as broadly applicable as conventional database systems, we would like to highlight the following challenges.

1. Apart from being uncertain, data can be annotated by other dimensions such as time or location. These techniques are partly already supported by traditional DBs, but to enable this kind of data in PDBs, we need to extend the probabilistic data models to support additional data dimensions.
2. Allowing a wide range of expressive queries, which can be executed efficiently, was one of the ingredients that made traditional database systems successful. Even though the query evaluation problem has been studied intensively in PDBs, for many classes of queries efficient ways of computing answers along with probabilities are not established yet.
3. Most importantly, the field of creating and updating PDBs still is in an early stage, where only very few initial results exist so far. Nevertheless, we believe that supporting the learning or updating of tuple probabilities from labeled training data and selective user inputs will be a key building block for future PDB approaches.

The remainder of this chapter thus is structured as follows. In Section 2, we establish the basic concepts and definitions known from relational databases which form also the basis for defining PDBs in Section 3. Next, in Section 4, we describe a closed and complete data model for both temporal and probabilistic databases (TPDBs), thus capturing data that is not only uncertain but also is annotated with time information. Section 5 discusses query evaluation in PDBs and describes an efficient top-$k$ style evaluation strategy in this context. Last, in

Section 6, we introduce the problem of learning tuple probabilities from labeled query answers, which allows also for updating and cleaning a PDB. Section 7 summarizes and concludes this chapter.

## 2 Relational Databases

The primary purpose of our first technical section is to establish the basic concepts and notations known from *relational databases*, which will form also the basis for the remainder of this chapter. We use a Datalog-oriented notation to represent intensional knowledge in the form of logical rules. Datalog thus fulfills two purposes in our setting. On the one hand, we employ Datalog to write *deduction rules*, from which we derive new intensional tuples from the existing database tuples for query answering. On the other hand, we also employ Datalog to encode *consistency constraints*, which allow us to remove inconsistent tuples from both the input relations and the query answers. For a broader background on the theoretical foundations of relational databases, including the relationship between Datalog, Relational Algebra and SQL, we refer the interested reader to one of the two standard references [2,34] in this field.

### 2.1 Relations and Tuples

We start with the two most basic concepts of relational databases, namely relations and tuples. We consider a *relation* $R$ as a logical predicate of arity $r \geq 1$. Together with a finite set of *attributes* $A_1, \ldots, A_m \in \mathcal{A}$ and a finite set of (potentially infinite) *domains* $\Omega_1, \ldots, \Omega_m \in \mathcal{O}$, we refer to $R(A_1, \ldots, A_r)$ also as the *schema* of relation $R$, where $dom : \mathcal{A} \to \mathcal{O}$ is a *domain mapping function* that maps the set of attributes onto their corresponding domains.

For a fixed universe of constants $\mathcal{U} = \bigcup_{\Omega_i \in \mathcal{O}} \Omega_i$, a *relation instance* $\mathcal{R}$ then is a finite subset $\mathcal{R} \subseteq \mathcal{U}^r$. We call the elements of $\mathcal{R}$ *tuples*, and we write $R(\bar{a})$ to denote a tuple in $\mathcal{R}$, where $\bar{a}$ is a vector of constants in $\mathcal{U}$. Furthermore, for a fixed set of variables $\mathcal{V}$, we use $R(\bar{X})$ to refer to a first-order literal over relation $R$, where $\bar{X} \subseteq \mathcal{U} \cup \mathcal{V}$ denotes a vector consisting of both variables and constants. We will use $Var(\bar{X}) \subseteq \mathcal{V}$ to refer to the set of variables in $\bar{X}$.

**Definition 1.** *Given relations $R_1, \ldots, R_n$, a relational database comprises the relation instances $\mathcal{R}_i$ whose tuples we collect in the single set of* extensional tuples $\mathcal{T} := \mathcal{R}_1 \cup \cdots \cup \mathcal{R}_n$.

In other words, a relation instance can simply be viewed as a table. A tuple thus denotes a row (or "record") in such a table. For convenience of notation, we collect the sets of tuples stored in all relation instances into a single set $\mathcal{T}$. In a deterministic database setting, we can thus say that a tuple $R(\bar{a})$ that is composed of a vector of constants in $\mathcal{U}$ is *true* iff $R(\bar{a}) \in \mathcal{T}$ (which we will also refer to as a "database tuple" in this case). As a shorthand notation, we will also employ $\mathcal{I} = \{I_1, \ldots, I_{|\mathcal{T}|}\}$ as a set of unique tuple identifiers.

*Example 1.* We consider a database with two relation instances from the movie domain, which capture information about the directors of movies and the awards that various movies may have won.

*Directed*

| | *Director* | *Movie* |
|---|---|---|
| $I_1$ | *Coppola* | *ApocalypseNow* |
| $I_2$ | *Coppola* | *Godfather* |
| $I_3$ | *Tarantino* | *PulpFiction* |

*WonAward*

| | *Movie* | *Award* |
|---|---|---|
| $I_4$ | *ApocalypseNow* | *BestScript* |
| $I_5$ | *Godfather* | *BestDirector* |
| $I_6$ | *Godfather* | *BestPicture* |
| $I_7$ | *PulpFiction* | *BestPicture* |

For example, the tuple *Directed*(*Coppola*, *ApocalypseNow*), which we also abbreviate by $I_1$, indicates that *Coppola* directed the movie *ApocalypseNow*. Thus, the above database contains two relation instances with tuples $\mathcal{T} = \{I_1, \ldots, I_7\}$.

### 2.2 Deduction Rules

To derive new tuples (and entire relations) from an existing relational database, we employ deduction rules. These can be viewed as generally applicable "if-then-rules". That is, given a condition, its conclusion follows. Formally, we follow Datalog [2,10] terminology but employ a more logic-oriented notation to express these rules. Each deduction rule takes the shape of a logical implication, with a conjunction of both positive and negative literals in the body (the "antecedent") and exactly one positive head literal (the "consequent"). Relations occurring in the head literal of a deduction rule are called *intensional relations* [2]. In contrast, relations holding the database tuples, i.e., those from $\mathcal{T}$, are also called *extensional relations*. These two sets of relations (and hence logical predicates) must not overlap and are used strictly differently within the deduction rules.

**Definition 2.** *A* deduction rule *is a logical rule of the form*

$$R(\bar{X}) \leftarrow \bigwedge_{i=1,\ldots,n} R_i(\bar{X}_i) \quad \wedge \bigwedge_{j=1,\ldots,m} \neg R_j(\bar{X}_j) \ \wedge \ \Phi(\bar{X}_A)$$

*where*

1. *$R$ denotes the intensional relation of the head literal, whereas $R_i$ and $R_j$ may refer to both intensional and extensional relations;*
2. *$n \geq 1$, $m \geq 0$, thus requiring at least one positive relational literal;*
3. *$\bar{X}$, $\bar{X}_i$, $\bar{X}_j$, and $\bar{X}_A$ denote tuples of both variables and constants, such that $Var(\bar{X}) \cup Var(\bar{X}_j) \cup Var(\bar{X}_A) \subseteq \bigcup_i Var(\bar{X}_i)$;*
4. *$\Phi(\bar{X}_A)$ is a conjunction of arithmetic predicates such as "$=$" and "$\neq$".*

*We refer to a set of deduction rules $\mathcal{D}$ also as a* Datalog program.

By the second condition of Definition 2, we require each deduction rule to have at least one positive literal in its body. Moreover, the third condition ensures *safe* deduction rules [2], by requiring that all variables in the head, $Var(\bar{X})$, in

negated literals, $Var(\bar{X}_j)$, and in arithmetic predicates, $Var(\bar{X}_A)$, occur also in at least one of the positive relational predicates, $Var(\bar{X}_i)$, in the body of each rule. As denoted by the fourth condition, we allow a conjunction of arithmetic comparisons such as "=" and "≠". All variables occurring in a deduction rule are implicitly universally quantified.

*Example 2.* Imagine we are interested in famous movie directors. To derive these from the tuples in Example 1, we can reason as follows: "if a director's movie won an award, then the director should be famous." As a logical formula, we express this as follows.

$$FamousDirector(X) \leftarrow Directed(X, Y) \wedge WonAward(Y, Z) \tag{3}$$

The above rule fulfills all requirements of Definition 2, since (1) all relations in the body are extensional, (2) there are two positive predicates, $n = 2$, and no negative predicate, $m = 0$, and (3) the single variable $X$ of the head is bound by a positive relational predicate in the body.

For the remainder of this chapter, we consider only *non-recursive* Datalog programs. Thus, our class of deduction rules coincides with the core operations that are expressible in Relational Algebra and in SQL [2], including *selections*, *projections*, and *joins*. All operations in Datalog (just like in Relation Algebra, but unlike SQL) eliminate duplicate tuples from the intensional relation instances they produce.

### 2.3 Grounding

The process of applying a deduction rule to a database instance, i.e., employing the rule to derive new tuples, is called *grounding*. In the next step, we thus explain how to instantiate the deduction rules, which we achieve by *successively substituting* the variables occurring a rule's body and head literals with constants occurring in the extensional relations and in other deduction rules [2,67].

**Definition 3.** *A* substitution $\sigma : \mathcal{V} \to \mathcal{V} \cup \mathcal{U}$ *is a mapping from variables* $\mathcal{V}$ *to variables and constants* $\mathcal{V} \cup \mathcal{U}$*. A substitution* $\sigma$ *is applied to a first-order formula* $\Phi$ *as follows:*

| *Definition* | *Condition* |
|---|---|
| $\sigma(\bigwedge_i \Phi_i) := \bigwedge_i \sigma(\Phi_i)$ | |
| $\sigma(\bigvee_i \Phi_i) := \bigvee_i \sigma(\Phi_i)$ | |
| $\sigma(\neg\Phi) := \neg\sigma(\Phi_i)$ | |
| $\sigma(R(\bar{X})) := R(\bar{Y})$ | $\sigma(\bar{X}) = \bar{Y}$ |

In general, substitutions can rename variables or replace variables by constants. If all variables are substituted by constants, then the resulting rule or literal is called *ground*.

*Example 3.* A valid substitution is given by $\sigma(X) = Coppola, \sigma(Y) = Godfather$, where we replace the variables $X$ and $Y$ by the constants *Coppola* and *Godfather*, respectively. If we apply the substitution to the deduction rule of Equation (3), we obtain

$$FamousDirector(Coppola) \leftarrow \begin{pmatrix} Directed(Coppola, Godfather) \\ \wedge\, WonAward(Godfather, Z) \end{pmatrix}$$

where $Z$ remains a variable.

We now collect all substitutions for a first-order deduction rule which are possible over a given database or a set of tuples to obtain a set of propositional formulas. These substitutions are called *ground rules* [2,10].

**Definition 4.** *Given a set of tuples $\mathcal{T}$ and a deduction rule $D$*

$$R(\bar{X}) \;\leftarrow \bigwedge_{i=1,\ldots,n} R_i(\bar{X}_i) \;\wedge \bigwedge_{j=1,\ldots,m} \neg R_j(\bar{X}_j) \;\wedge\; \Phi(\bar{X}_A)$$

*the* ground rules $G(D, \mathcal{T})$ *are all substitutions $\sigma$ where*

1. *$\sigma$'s preimage coincides with $\bigcup_i Var(\bar{X}_i)$;*
2. *$\sigma$'s image consists of constants only;*
3. *$\forall i : \sigma(R_i(\bar{X}_i)) \in \mathcal{T}$;*
4. *$\sigma(\Phi(\bar{X}_A)) \equiv true$.*

The first and second condition requires the substitution to bind all variables in the deduction rule to constants. In addition, all positive ground literals have to match a tuple in $\mathcal{T}$. In the case of a deterministic database, negated literals must not match any tuple. Later, in a probabilistic database context, however, they may indeed match a tuple, which is why we omit a condition on this case. The last condition ensures that the arithmetic literals are satisfied.

*Example 4.* Let the deduction rule of Equation (3) be $D$. For the tuples of Example 1, there are four substitutions $G(D, \{I_1, \ldots, I_7\}) = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$, where:

$$\begin{array}{ll} \sigma_1(X) = Coppola & \sigma_2(X) = Coppola \\ \sigma_1(Y) = ApocalypseNow & \sigma_2(Y) = Godfather \\ \sigma_1(Z) = BestScript & \sigma_2(Z) = BestDirector \\ \\ \sigma_3(X) = Coppola & \sigma_4(X) = Tarantino \\ \sigma_3(Y) = Godfather & \sigma_4(Y) = PulpFiction \\ \sigma_3(Z) = BestPicture & \sigma_4(Z) = BestPicture \end{array}$$

All substitutions provide valid ground rules according to Definition 4, because (1) their preimages coincide with all variables of $D$, (2) their images are constants only, (4) there are no arithmetic literals, and (3) all positive body literals match the following database tuples:

| Literal | Tuple | Literal | Tuple |
|---|---|---|---|
| $\sigma_1(Directed(X,Y))$ | $I_1$ | $\sigma_1(WonAward(Y,Z))$ | $I_4$ |
| $\sigma_2(Directed(X,Y))$ | $I_2$ | $\sigma_2(WonAward(Y,Z))$ | $I_5$ |
| $\sigma_3(Directed(X,Y))$ | $I_2$ | $\sigma_3(WonAward(Y,Z))$ | $I_6$ |
| $\sigma_4(Directed(X,Y))$ | $I_3$ | $\sigma_4(WonAward(Y,Z))$ | $I_7$ |

Finally, we employ the groundings of a deduction rule to derive new tuples by instantiating the head literal of the rule.

**Definition 5.** *Given a single deduction rule* $D := (R(\bar{X}) \leftarrow \Psi) \in \mathcal{D}$ *and a set of extensional tuples* $\mathcal{T}$*, the* intensional tuples *are created as follows:*

$$\mathit{IntensionalTuples}(D, \mathcal{T}) := \{\sigma(R(\bar{X})) \mid \sigma \in G(D, \mathcal{T})\}$$

We note that the same new tuple might result from more than one substitution, as it is illustrated by the following example.

*Example 5.* Let $D$ be the deduction rule of Equation (3). Continuing Example 4, there are two new intensional tuples:

$$\mathit{IntensionalTuples}(D, \{I_1, \ldots, I_7\}) = \begin{Bmatrix} \mathit{FamousDirector}(\mathit{Coppola}), \\ \mathit{FamousDirector}(\mathit{Tarantino}) \end{Bmatrix}$$

The first tuple originates from $\sigma_1$, $\sigma_2$, $\sigma_3$ of Example 4, whereas the second tuple results only from $\sigma_4$.

### 2.4 Queries and Query Answers

We now move on to define queries and their answers over a relational database with deduction rules. Just like the antecedents of the deduction rules, our queries consist of conjunctions of both positive and negative literals.

**Definition 6.** *Given a set of deduction rules* $\mathcal{D}$*, which define our intensional relations, a* query $Q$ *is a conjunction:*

$$Q(\bar{X}) \;:=\; \bigwedge_{i=1,\ldots,n} R_i(\bar{X}_i) \;\; \wedge \bigwedge_{j=1,\ldots,m} \neg R_j(\bar{X}_j) \;\wedge\; \Phi(\bar{X}_A)$$

*where*

1. *all* $R_i$*,* $R_j$ *are intensional relations in* $\mathcal{D}$*;*
2. $\bar{X}$ *are called* query variables *and it holds that* $\mathit{Var}(\bar{X}) = \bigcup_{i=1,\ldots,n} \mathit{Var}(\bar{X}_i)$*;*
3. *all variables in negated or arithmetic literals are bound by positive literals such that* $\mathit{Var}(\bar{X}_A) \subseteq \bigcup_{i=1,\ldots,n} \mathit{Var}(\bar{X}_i)$*, and for all* $j \in \{1, \ldots, m\}$ *it holds that* $\mathit{Var}(\bar{X}_j) \subseteq \bigcup_{i=1,\ldots,n} \mathit{Var}(\bar{X}_i)$*;*
4. $\Phi(\bar{X}_A)$ *is a conjunction of arithmetic predicates such as "=" and "*$\neq$*".*

The first condition allows us to ask for head literals of any deduction rule. The set of variables in positive literals are precisely the query variables. The final two conditions ensure safeness as in deduction rules. We want to remark that for a theoretical analysis, it suffices to have only one intensional literal as a query, since the deduction rules allow us to encode any combination of relational operations such as projections, selections or joins. However, for practical purposes, it is often useful to combine more than one literal into a query via a conjunction.

*Example 6.* Extending Examples 1 and 2, we can formulate the query

$$Q(X) := \mathit{FamousDirector}(X) \wedge (X \neq \mathit{Tarantino})$$

which asks for all famous directors except *Tarantino*. Thus, the only query variable in this example is $X$.

Since queries have the same shape as the antecedents of the deduction rules, we apply Definition 4 also for grounding the queries. Assuming that $\mathcal{T}'$ comprises all database tuples and all new intensional tuples resulting from grounding the deduction rules, we may again rely on $G(Q(\bar{X}), \mathcal{T}')$ to define the query answers.

**Definition 7.** *For a set of tuples $\mathcal{T}'$ and a query $Q(\bar{X})$, the set of* query answers *is given by:*

$$\mathit{QueryAnswers}(Q(\bar{X}), \mathcal{T}') := \{\sigma(Q(\bar{X})) \mid \sigma \in G(Q(\bar{X}), \mathcal{T}')\}$$

Thus, each answer provides a distinct binding of (all of its) query variables to constants in $\mathcal{U}$.

*Example 7.* For the query $Q(X)$ of Example 6 and the deduction rule of Example 2, there exists only one answer, namely *FamousDirector*(*Coppola*).

Again, in a deterministic database setting, we can thus say that a tuple $R(\bar{a})$ (which may now refer to either a "database tuple" or a "derived tuple") is *true* iff $R(\bar{a}) \in \mathcal{T}'$. This assumption will be relaxed in the next section.

## 3 Probabilistic Databases

We now move on to present a model for probabilistic databases. This model extends the one for relational databases by using probabilities.

### 3.1 Possible Worlds

In this subsection, we relax the common assumption in deterministic databases, namely that all tuples, which are captured in both the extensional and intensional relations of the database, are certainly *true*. Depending on the existence (i.e., the "correctness") of the tuples, a database can be in different states. Each such state is called a *possible world* [3,65].

**Definition 8.** *For a relational database with extensional tuples $\mathcal{T}$, a* possible world *is a subset $\mathcal{W} \subseteq \mathcal{T}$.*

The interpretation of a possible world is as follows. All tuples in $\mathcal{W}$ exist (i.e., they are *true* in $\mathcal{W}$), whereas all tuples in $\mathcal{T} \backslash \mathcal{W}$ do not exist (i.e., they are *false* in $\mathcal{W}$). In the absence of any constraints that would restrict this set of possible worlds (see Subsection 3.6), any subset $\mathcal{W}$ of tuples in $\mathcal{T}$ forms a valid possible world (aka. "possible instance") of the probabilistic database. Hence, there are $2^{|\mathcal{T}|}$ possible worlds.

*Example 8.* Considering the relational database of Example 1, a possible world is $\mathcal{W} := \{I_2, I_4, I_6\}$, which hence has only one tuple in the *Directed* relation and two tuples in the *WonAward* relation.

### 3.2 Probabilistic Database Model

Based on the possible worlds semantics, we can now formally introduce probabilistic databases [65], which—in their most general form—simply impose a probability distribution over the set of possible worlds.

**Definition 9.** *Given a set of tuples $\mathcal{T}$ with possible worlds $\mathcal{W}_1, \ldots, \mathcal{W}_n$, a* probabilistic database (PDB) *assigns a probability $P : 2^{\mathcal{T}} \to [0, 1]$ to each possible world $\mathcal{W} \subseteq \mathcal{T}$, such that:*

$$\sum_{\mathcal{W} \subseteq \mathcal{T}} P(\mathcal{W}) = 1$$

In other words, in a PDB the probabilities of the possible worlds $P(\mathcal{W})$ form a probability distribution. Thus, each possible world can be seen as the outcome of a probabilistic experiment.

*Example 9.* If we allow only two possible worlds $\mathcal{W}_1 := \{I_1, I_3, I_5, I_7\}$ and $\mathcal{W}_2 := \{I_2, I_4, I_6\}$ over the tuples of Example 1, we can set their probabilities to $P(\mathcal{W}_1) = 0.4$ and $P(\mathcal{W}_2) = 0.6$ to obtain a valid PDB.

We remark that the above possible-worlds semantics, which is the predominant data model of virtually any recent PDB approach [65], is a very expressive representation formalism for probabilistic data. By defining a probability distribution over the possible instances of the underlying deterministic database, it in principle allows us to represent any form of correlation among the extensional tuples. In practice, however, it is usually not permissible to store an exponential amount of possible worlds over the set of extensional tuples $\mathcal{T}$. We thus now move on to the concept of *tuple independence*.

### 3.3 Tuple Independence

Since there are exponentially many possible worlds, it is prohibitive to store every possible world along with its probability in an actual database system. Instead, we opt for a simpler method by annotating each individual tuple with a probability value. By assuming that the probabilities of all tuples are independent [27,60], we obtain the representation model of *tuple-independent PDBs* [15,65].

**Definition 10.** *For a set of extensional tuples $\mathcal{T}$, a* tuple-independent PDB *$(\mathcal{T}, p)$ is a pair, where*

1. *$p$ is a function $p : \mathcal{T} \to (0, 1]$, which assigns a non-zero* probability value *$p(I)$ to each tuple $I \in \mathcal{T}$;*
2. *the probability values of all tuples in $\mathcal{T}$ are assumed to be* independent*;*
3. *every subset $\mathcal{W} \subseteq \mathcal{T}$ is a possible world and has probability:*

$$P(\mathcal{W}, \mathcal{T}) := \prod_{I \in \mathcal{W}} p(I) \quad \cdot \prod_{I \in \mathcal{T} \setminus \mathcal{W}} (1 - p(I))$$

The probability $p(I)$ of a tuple $I$ denotes the confidence in the existence of the tuple in the database where a higher value $p(I)$ denotes a higher confidence in $I$ being valid. However, the probabilities of different tuples do not depend on each other; that is, they are assumed to be probabilistically *independent*. This allows us to multiply the probabilities of the tuples to obtain the probability of the possible world. From a probabilistic perspective, each extensional tuple corresponds to an independent binary random variable.

*Example 10.* Assuming we are unsure about the existence of each of the tuples in Example 1, we may now annotate them with probabilities as follows.

*Directed*

| | *Director* | *Movie* | $p$ |
|---|---|---|---|
| $I_1$ | *Coppola* | *ApocalypseNow* | 0.7 |
| $I_2$ | *Coppola* | *Godfather* | 0.5 |
| $I_3$ | *Tarantino* | *PulpFiction* | 0.2 |

*WonAward*

| | *Movie* | *Award* | $p$ |
|---|---|---|---|
| $I_4$ | *ApocalypseNow* | *BestScript* | 0.1 |
| $I_5$ | *Godfather* | *BestDirector* | 0.8 |
| $I_6$ | *Godfather* | *BestPicture* | 0.9 |
| $I_7$ | *PulpFiction* | *BestPicture* | 0.5 |

Here, *Coppola* directed the movie *Godfather* only with probability 0.5. In addition, the possible world $\mathcal{W} := \{I_1, I_3, I_5, I_7\}$ has the probability:

$$P(\mathcal{W}, \{I_1, \ldots, I_9\}) = 0.7 \cdot (1 - 0.5) \cdot 0.2 \cdot (1 - 0.1) \cdot 0.8 \cdot (1 - 0.9) \cdot 0.5 = 0.00252$$

In Subsection 3.2, we required a PDB to form a probability distribution over its possible worlds. For a tuple-independent PDB, we can now prove that this condition also holds.

**Proposition 1.** *Given a tuple-independent PDB $(\mathcal{T}, p)$, then $P(\mathcal{W}, \mathcal{T})$ of Definition 10 forms a probability distribution over the possible worlds $\mathcal{W} \subseteq \mathcal{T}$, such that:*

$$\sum_{\mathcal{W} \subseteq \mathcal{T}} P(\mathcal{W}, \mathcal{T}) = 1$$

*Proof.* We prove the proposition by induction over the cardinality of $\mathcal{T}$.

Basis i = 1:
$\sum_{\mathcal{W} \subseteq \{I_1\}} P(\mathcal{W}, \{I_1\}) = p(I_1) + (1 - p(I_1)) = 1$

Step $(i-1) \to i$:
Let $\mathcal{T} := \{I_1, \ldots, I_i\}$ where $I_i$ is the new tuple.

$$\begin{aligned}
&\textstyle\sum_{\mathcal{W} \subseteq \mathcal{T}} P(\mathcal{W}, \mathcal{T}) \\
&\quad = \textstyle\sum_{\mathcal{W} \subseteq \mathcal{T}} \prod_{I \in \mathcal{W}} p(I) \cdot \prod_{I \in \mathcal{T} \setminus \mathcal{W}} (1 - p(I)) \\
&\quad = \underbrace{(p(I_i) + (1 - p(I_i)))}_{=1} \cdot \underbrace{\textstyle\sum_{\mathcal{W} \subseteq \mathcal{T} \setminus \{I_i\}} \prod_{I \in \mathcal{W}} p(I) \cdot \prod_{I \in \mathcal{T} \setminus \mathcal{W}} (1 - p(I))}_{=1 \text{ by hypothesis}}
\end{aligned}$$

In the remaining parts of this chapter, we will always consider a tuple-independent PDB when we refer to a PDB.

### 3.4 Propositional Lineage

In this subsection, we introduce how to trace the derivation history of intensional tuples. In database terminology, this concept is commonly referred to as *data lineage* [7,12,56], which we represent via propositional (Boolean) formulas. More specifically, lineage relates each newly derived tuple in $\mathcal{T}'\backslash\mathcal{T}$ with the extensional tuples in $\mathcal{T}$ via the three Boolean connectives $\wedge$, $\vee$ and $\neg$, which reflect the semantics of the relational operations that were applied to derive that tuple.

**Definition 11.** *We establish* lineage *inductively via the function*

$$\lambda : \mathit{GroundLiterals} \rightarrow \mathit{Lineage}$$

*which is defined as follows:*

1. *For tuples $\mathcal{T}$ and $R(\bar{a})$ with $R$ being extensional and $R(\bar{a}) \in \mathcal{T}$, we have*

$$\lambda(R(\bar{a})) := I$$

   *where $I$ is a Boolean (random) variable representing the tuple $R(\bar{a})$.*
2. *For tuples $\mathcal{T}$, deduction rules $\mathcal{D}$, and $R(\bar{a})$ with $R$ being intensional, lineage is defined as*

$$\lambda(R(\bar{a})) := \bigvee_{\substack{D\in\mathcal{D},\\ \sigma\in G(D,\mathcal{T}),\\ \sigma(\bar{X})=\bar{a}}} \left( \bigwedge_{i=1,\ldots,n} \lambda(\sigma(R_i(\bar{X}_i))) \quad \wedge \bigwedge_{\sigma(R_j(\bar{X}_j))\in\mathcal{T}} \neg\lambda(\sigma(R_j(\bar{X}_j))) \right)$$

   *where $D$ is a deduction rule having $R$ as its head literal:*

$$R(\bar{X}) \leftarrow \bigwedge_{i=1,\ldots,n} R_i(\bar{X}_i) \quad \wedge \bigwedge_{j=1,\ldots,m} \neg R_j(\bar{X}_j) \ \wedge \ \Phi(\bar{X})$$

3. *If there is no match to $R(\bar{a})$ in both $\mathcal{T}$ and $\mathcal{D}$:*

$$\lambda(R(\bar{a})) := \mathit{false}$$

In the first case, we simply replace a ground literal $R(\bar{a})$ by a Boolean random variable $I$ that represents this database tuple. The second case however is slightly more involved. The ground literal $R(\bar{a})$ is replaced by the disjunction over all deduction rules and all groundings of thereof, where the ground head literal matched $R(\bar{a})$. Likewise, negative literals are only traced if they occur in the tuples. In the third case, all literals not being matched at all are replaced by the constant *false*, which resembles a *closed world assumption* that is common in databases and is known as "negation-as-failure" in Datalog [2]. Finally, arithmetic literals do not occur in the lineage formulas, since a successful grounding replaces them with the constant *true* (see Definition 4). Similarly, because a query has the same shape as the body of a deduction rule, we write $\lambda(Q(\bar{a}))$ to refer to the lineage formula associated with a query answer.

*Example 11.* Building on Examples 4 and 5, we determine the lineage of the tuple *FamousDirector*(*Coppola*), which was produced by the three substitutions $\sigma_1$, $\sigma_2$, and $\sigma_3$. The second case of Definition 11 delivers a disjunction ranging over both substitutions:

$$\begin{aligned}
&\lambda(\mathit{FamousDirector}(\mathit{Coppola})) = \\
&\quad \begin{pmatrix} \lambda(\mathit{Directed}(\mathit{Coppola}, \mathit{ApocalypseNow})) \\ \wedge\ \lambda(\mathit{WonAward}(\mathit{ApocalypseNow}, \mathit{BestScript})) \end{pmatrix} && \text{from } \sigma_1 \\
&\quad \vee \\
&\quad \begin{pmatrix} \lambda(\mathit{Directed}(\mathit{Coppola}, \mathit{Godfather})) \\ \wedge\ \lambda(\mathit{WonAward}(\mathit{Godfather}, \mathit{BestDirector})) \end{pmatrix} && \text{from } \sigma_2 \\
&\quad \vee \\
&\quad \begin{pmatrix} \lambda(\mathit{Directed}(\mathit{Coppola}, \mathit{Godfather})) \\ \wedge\ \lambda(\mathit{WonAward}(\mathit{Godfather}, \mathit{BestPicture})) \end{pmatrix} && \text{from } \sigma_3
\end{aligned}$$

Then, the first case of Definition 11 replaces all ground literals by their tuple identifiers:

$$\underbrace{(I_1 \wedge I_4)}_{\text{from } \sigma_1} \vee \underbrace{(I_2 \wedge I_5)}_{\text{from } \sigma_2} \vee \underbrace{(I_2 \wedge I_6)}_{\text{from } \sigma_3}$$

Next, we study the computational complexity of lineage tracing. It is known that grounding non-recursive Datalog rules, which coincides with our class of deduction rules, has polynomial data complexity [41]. Now, we extend this result to lineage tracing.

**Lemma 1.** *For a fixed set of deduction rules $\mathcal{D}$, grounding with lineage as of Definition 11 has polynomial data complexity in $|\mathcal{T}|$.*

*Proof.* We have to show that, according to Definition 11, lineage creates an overhead which is polynomial in $|\mathcal{T}|$. In the first and third case of the definition, we can see that we solely rely on a look-up in $\mathcal{D}$ or $\mathcal{T}$, which is computable in polynomial time. The second case iterates over all deduction rules $D \in \mathcal{D}$. For each deduction rule $D$, it performs a number of look-ups which is upper-bounded by $|G(D, \mathcal{T})| \cdot |D|$. Since grounding has polynomial data complexity, $G(D, \mathcal{T})$ is of polynomial size in $\mathcal{T}$. Thus, also the third case has polynomial data complexity.

We next introduce a normal form of propositional lineage formulas, which is very common in logic [62]. Assuming lineage formulas to be in a normal form will simplify proofs that follow later on.

**Definition 12.** *A propositional lineage formula $\phi$ is in* Disjunctive Normal Form *(DNF) if $\phi = \psi_1 \vee \cdots \vee \psi_n$ and each clause $\psi_i$ is of the form $\bigwedge_j I_j \ \wedge \bigwedge_k \neg I_k$.*

As an illustration, the lineage formula of Example 11 is in DNF. In general, any propositional formula can be transformed into DNF [62], which we rely on in order to show the following statement.

**Proposition 2.** *The deduction rules of Definition 2 allow us to express any propositional lineage formula.*

*Proof.* Consider a probabilistic database $(\mathcal{T}, p)$ and an arbitrary propositional formula $\phi$ connecting tuple identifiers. Without loss of generality, let the formula $\phi$ be in DNF and range over only one relation $R$. First, we introduce one additional tuple $R(0)$ and set $p(R(0)) = 1$. Then, for each clause $\psi_i = \bigwedge_j I_j \wedge \bigwedge_k \neg I_k$ of $\phi$, we create exactly one deduction rule:

$$R'(0) \leftarrow R(0) \wedge \bigwedge_j R(j) \wedge \bigwedge_k \neg R(k)$$

The lineage formula of the intensional tuple $R'(0)$ thus is $\phi$. The reason is that each rule creates one clause. Then, these clauses are connected by a disjunction that originates from the second case of Definition 11.

From the above consideration, it follows that the lineage formulas considered in our context may take more general forms than lineage formulas resulting from (unions of) conjunctive queries (UCQs) [14,16], which produce only formulas which are restricted to positive literals.

### 3.5 Computing Probabilities

Since in a probabilistic database each tuple exists only with a given probability, we can now quantify the probability that each answer exists. Based on [30,56,65], we compute probabilities of query answers via their lineage formulas. To achieve this, we interpret the propositional lineage formulas over a possible world of a probabilistic database $(\mathcal{T}, p)$ as follows. We say that a possible world $\mathcal{W}$ is a *model* [67] for a propositional lineage formula $\phi$, denoted as $\mathcal{W} \models \phi$, if, by setting all tuples in $\mathcal{W}$ to *true* and all tuples in $\mathcal{T} \backslash \mathcal{W}$ to *false*, $\mathcal{W}$ represents a truth assignment that satisfies $\phi$. Moreover, let the set $\mathcal{M}(\phi, \mathcal{T})$ contain all possible worlds $\mathcal{W} \subseteq \mathcal{T}$ being a model for a propositional lineage formula $\phi$.

$$\mathcal{M}(\phi, \mathcal{T}) := \{\mathcal{W} \mid \mathcal{W} \subseteq \mathcal{T}, \mathcal{W} \models \phi\} \tag{4}$$

If it is clear from the context, we drop $\mathcal{T}$ as an argument of $\mathcal{M}$. We compute the *probability* of any Boolean formula $\phi$ over tuples in $\mathcal{T}$ as the sum of the probabilities of all the possible worlds that are a model for $\phi$:

$$P(\phi) := \sum_{\mathcal{W} \in \mathcal{M}(\phi, \mathcal{T})} P(\mathcal{W}, \mathcal{T}) \tag{5}$$

Here, $P(\mathcal{W}, \mathcal{T})$ is as in Definition 10. We can interpret the above probability as the marginal probability of the lineage formula $\phi$. The above sum can range over exponentially many terms. However, in practice, we can—at least in many cases—compute the probability $P(\phi)$ directly via the structure of the lineage

formula $\phi$. Let $\mathit{Tup}(\phi) \subseteq \mathcal{T}$ denote the set of tuples occurring in $\phi$. Then, the following computations can be employed:

| Definition | Condition |
|---|---|
| $P(I) := p(I)$ | $I \in \mathcal{T}$ |
| $P(\bigwedge_i \phi_i) := \prod_i P(\phi_i)$ | $i \neq j \Rightarrow \mathit{Tup}(\phi_i) \cap \mathit{Tup}(\phi_j) = \emptyset$ |
| $P(\bigvee_i \phi_i) := 1 - \prod_i (1 - P(\phi_i))$ | $i \neq j \Rightarrow \mathit{Tup}(\phi_i) \cap \mathit{Tup}(\phi_j) = \emptyset$ |
| $P(\phi \vee \psi) := P(\phi) + P(\psi)$ | $\phi \wedge \psi \equiv \mathit{false}$ |
| $P(\neg\phi) := 1 - P(\phi)$ | |
| $P(\mathit{true}) := 1$ | |
| $P(\mathit{false}) := 0$ | |

(6)

The first line captures the case of an extensional tuple $I$, for which we return its attached probability value $p(I)$. The next two lines handle *independent-and* and *independent-or* operations for conjunctions and disjunctions over tuple-disjoint subformulas $\phi_i$, respectively. In the fourth line, we address disjunctions for subformulas $\phi$ and $\psi$ that denote disjoint probabilistic events (*disjoint-or*). The fifth line handles negation. Finally, the probability of *true* and *false* is 1 and 0, respectively.

*Example 12.* Let us compute the probability $P(I_1 \wedge I_2 \wedge \neg I_3)$ over the tuples of Example 10. First, the second line of Equation (6) is applicable, which yields $P(I_1) \cdot P(I_2) \cdot P(\neg I_3)$. Next, we can replace the negation to obtain $P(I_1) \cdot P(I_2) \cdot (1 - P(I_3))$. Now, looking up the tuples' probability values in Example 10 yields $0.7 \cdot 0.5 \cdot (1 - 0.2) = 0.28$.

The definition of $P(\phi)$ presented in Equation (6) can be evaluated in linear time in the size of $\phi$. However, for general lineage formulas, computing $P(\phi)$ is known to be $\#\mathcal{P}$-hard [16,15,47]. Here, $\#\mathcal{P}$ [68] denotes a class of counting problems. Its prototypical problem, $\#\mathcal{SAT}$, asks for the number of satisfying assignments of a propositional formula and may thus have to consider a number of satisfying assignment that is exponential in the number of variables in the formula.

We next present a number of deduction rules which are known to yield lineage formulas that may exhibit computationally hard instances in terms of probability computations.

**Lemma 2.** *Let a probabilistic database $(\mathcal{T}, p)$ and the following deduction rules be given:*

$$H(0) \leftarrow R(X) \wedge S(X,Y) \wedge T(Y)$$

$$H(1) \leftarrow R(X) \wedge S(X,Y)$$
$$H(1) \leftarrow S(X,Y) \wedge T(Y)$$

$$H(2) \leftarrow R(X) \wedge S_1(X,Y)$$
$$H(2) \leftarrow S_1(X,Y) \wedge S_2(X,Y)$$
$$H(2) \leftarrow S_2(X,Y) \wedge T(Y)$$

$$H(3) \leftarrow R(X) \wedge S_1(X,Y)$$
$$H(3) \leftarrow S_1(X,Y) \wedge S_2(X,Y)$$
$$H(3) \leftarrow S_2(X,Y) \wedge S_3(X,Y)$$
$$H(3) \leftarrow S_3(X,Y) \wedge T(Y)$$
$$\ldots$$

*Then, for each $H(k)$ the corresponding computations of the probabilities $P(\lambda(H_k))$ are $\#\mathcal{P}$-hard in $|\mathcal{T}|$.*

In the lemma above, $k$ is a constant, hence $H(0)$ is a ground literal resembling a Boolean query. A formal proof for the above statement can be found in [17].

To be able to address also these hard cases, we employ the following equation, called *Shannon expansion*, which is applicable to any propositional lineage formula:

$$P(\phi) := p(I) \cdot P(\phi_{[I/true]}) + (1 - p(I)) \cdot P(\phi_{[I/false]}) \tag{7}$$

Here, the notation $\phi_{[I/true]}$ for a tuple $I \in Tup(\phi)$ denotes that we replace all occurrences of $I$ in $\phi$ by *true*. Shannon expansion is based on the following logical equivalence:

$$\phi \equiv (I \wedge \phi_{[I/true]}) \vee (\neg I \wedge \phi_{[I/false]}) \tag{8}$$

The resulting disjunction fulfills the *disjoint-or* condition (see Equation (6)) with respect to $I$. Repeated applications of Shannon expansions may however increase the size of $\phi$ exponentially, and hence do not circumvent the computational hardness of the problem.

*Example 13.* We calculate the probability of the lineage formula of Example 11 as follows:

$$P((I_1 \wedge I_4) \vee (I_2 \wedge I_5) \vee (I_2 \wedge I_6))$$

The top-level operator is a disjunction where the third line of Equation (6) is not applicable, since $I_2$ occurs in two subformulas. Hence, we first apply a Shannon expansion for $I_2$:

$$p(I_2) \cdot P((I_1 \wedge I_4) \vee I_5 \vee I_6) + (1 - p(I_2)) \cdot P(I_1 \wedge I_4)$$

Now, we can resolve the disjunction and the conjunction by independent-or and independent-and, respectively:

$$p(I_2) \cdot (1 - (1 - p(I_1) \cdot p(I_4)) \cdot (1 - p(I_5)) \cdot (1 - p(I_6))) + (1 - p(I_2)) \cdot p(I_1) \cdot p(I_4)$$

**Partial Derivatives.** As introduced in [40,52], we can quantify the impact of the probability of a tuple $p(I)$ on the probability $P(\phi)$ of a propositional lineage formula $\phi$ by its partial derivative, which has many applications to sensitivity analysis [40] and gradient-based optimization methods [45] (see also Section 6.4).

**Definition 13.** *Given a propositional lineage formula $\phi$ and a tuple $I \in Tup(\phi)$, the* partial derivative *of $P(\phi)$ with respect to $p(I)$ is*

$$\frac{\partial P(\phi)}{\partial p(I)} := \frac{P(\phi_{[I/true]}) - P(\phi_{[I/false]})}{P(true) - P(false)} = P(\phi_{[I/true]}) - P(\phi_{[I/false]})$$

Again, $\phi_{[I/true]}$ means that all occurrences of $I$ in $\phi$ are replaced by *true* (and analogously for *false*).

*Example 14.* We may determine the derivative of the probability of the propositional lineage formula $\phi := I_1 \wedge I_4$ with respect to the tuple $I_4$ as follows:

$$\begin{aligned} \tfrac{\partial P(\phi)}{\partial p(I_4)} &= P((I_1 \wedge I_4)_{[I_4/true]}) - P((I_1 \wedge I_4)_{[I_4/false]}) \\ &= p(I_1) - P(false) \\ &= p(I_1) \end{aligned}$$

### 3.6 Consistency Constraints

To rule out instances (i.e., possible worlds) of the probabilistic database, which would be inconsistent with assumptions we may make about the real world, we support *consistency constraints*. For instance, if for the same person two places of birth are stored in the database, then we may intend to remove one of them by a consistency constraint. In general, we consider the constraints to be presented in the form of a single propositional lineage formula $\phi_c$, which connects different tuple identifiers. Intuitively, the constraint formula $\phi_c$ describes all possible worlds that are valid. In contrast, all possible worlds that do not satisfy the constraint will be dropped from the probability computations. Because it is tedious to manually formulate a propositional formula over many database tuples, we allow $\phi_c$ to be induced by deduction rules $\mathcal{D}_c$ and two sets of queries $\mathcal{C}_p$ and $\mathcal{C}_n$ as follows. For simplicity, we assume $\mathcal{C}_p \cap \mathcal{C}_n = \emptyset$ and $\mathcal{D}_c \cap \mathcal{D}_q = \emptyset$, where $\mathcal{D}_q$ are the deduction rules related to the query.

**Definition 14.** *Let a set of deduction rules $\mathcal{D}_c$ and two sets $\mathcal{C}_p$ and $\mathcal{C}_n$ of intensional literals from $\mathcal{D}_c$ be given. If $\mathcal{T}$ contains all tuples deducible by $\mathcal{D}_c$, then the constraint formula $\phi_c$ is obtained by:*

$$\phi_c := \bigwedge_{\substack{C_p(\bar{X}) \in \mathcal{C}_p, \\ C_p(\bar{a}) \in Answers(C_p(\bar{X}), \mathcal{T})}} \lambda(C_p(\bar{a})) \quad \wedge \quad \bigwedge_{\substack{C_p(\bar{X}) \in \mathcal{C}_n, \\ C_n(\bar{a}) \in Answers(C_n(\bar{X}), \mathcal{T})}} \neg\lambda(C_n(\bar{a}))$$

Hence, based on the above definition, we create constraints on probabilistic databases directly via deduction rules. All answers from literals in $\mathcal{C}_p$ yield propositional lineage formulas which must always hold, whereas the lineage formulas being derived from literals in $\mathcal{C}_n$ must never hold. We connect all these ground constraints, i.e., their lineage formulas, by a conjunction to enforce all of them together. It is important to note that the deduction rules of the constraints do not create any new tuples, but merely serve the purpose of creating the propositional constraint formula $\phi_c$.

*Example 15.* Let us formalize that every movie is directed by only one person. Suppose we create the following deduction rule

$$\mathit{Constraint}(P_1, P_2, M) \leftarrow (\mathit{Directed}(P_1, M) \wedge \mathit{Directed}(P_2, M) \wedge P_1 \neq P_2)$$

and insert $\mathit{Constraint}(P_1, P_2, M)$ into $\mathcal{C}_n$, which hence disallows the existence of two persons $P_1$ and $P_2$ that both directed the same movie.

Due to the logical implication, we may also abbreviate constraints consisting of a single deduction rule by the body of the deduction rule only. That is, we may just omit the head literal in these cases.

*Example 16.* We can write the constraint of Example 15 without the head literal as follows:

$$\neg(\mathit{Directed}(P_1, M) \wedge \mathit{Directed}(P_2, M) \wedge P_1 \neq P_2)$$

Here, the negation indicates that the former head literal was in $\mathcal{C}_n$.

With respect to the probability computations, constraints remove all the possible worlds from the computations, which violate the constraint. This process is called *conditioning* [43], which can be formally defined as follows.

**Definition 15.** *Let constraints be given as a propositional lineage formula $\phi_c$ over a probabilistic database $(\mathcal{T}, p)$. If $\phi_c$ is satisfiable, then the probability $P(\psi)$ of a propositional lineage formula $\psi$ over $\mathcal{T}$ can be* conditioned *onto $\phi_c$ as follows:*

$$P(\psi \mid \phi_c) := \frac{P(\psi \wedge \phi_c)}{P(\phi_c)} \tag{9}$$

In the above definition, $\psi$ can represent any lineage formula, in particular also that of a query answer. After removing the possible worlds violating a constraint from the probabilistic database, conditioning (re-)weights the remaining worlds such that they again form a probability distribution.

*Example 17.* We consider the lineage formula $\psi = I_2 \wedge (I_5 \vee I_6)$ over the tuples of Example 10. Without any constraints, its probability is computed by Equation (6) as $P(\psi) = 0.5 \cdot (1 - (1 - 0.8) \cdot (1 - 0.9)) = 0.49$. If we set $\phi_c = I_2$ as a constraint, we remove all possible worlds that exclude $I_2$. Consequently, the probability is updated to:

$$P(\psi \mid I_2) = \frac{P(I_2 \wedge (I_5 \vee I_6))}{P(I_2)} = \frac{p(I_2) \cdot P(I_5 \vee I_6)}{p(I_2)} = P(I_5 \vee I_6) = 0.98$$

In the following, we characterize a useful property of constraints. If a number of constraints do not share any tuple with a lineage formula $\psi$, then the probability $P(\psi)$ is not affected by the constraints.

**Proposition 3.** *If the constraints $\phi_c$ and the lineage formula $\psi$ are independent with respect to their database tuples, i.e., $\mathit{Tup}(\psi) \cap \mathit{Tup}(\phi_c) = \emptyset$, then it holds that:*

$$P(\psi \mid \phi_c) = P(\psi)$$

*Proof.* Due to the second line of Equation (6) and $Tup(\psi) \cap Tup(\phi_c) = \emptyset$, we can write $P(\psi \wedge \phi_c) = P(\psi) \cdot P(\phi_c)$ . Therefore, the following equation holds:

$$P(\psi \mid \phi_c) = \frac{P(\psi \wedge \phi_c)}{P(\phi_c)} = P(\psi) \cdot \frac{P(\phi_c)}{P(\phi_c)} = P(\psi)$$

Hence, if we have the constraint $\phi_c \equiv \mathit{true}$, the standard unconditioned probability computations of Section 3.5 arise as a special case. Finally, since Equation (9) invokes probability computations on the constraint $\phi_c$, constraints may also yield $\#\mathcal{P}$-hard computations, which we capture next.

**Observation 11** *Constraints can cause $\#\mathcal{P}$-hard probability computations.*

The reason is that one of the lineage formulas described in Lemma 2 could occur in $\phi_c$.

**Expressiveness of Constraints.** The deduction rules of Definition 14, which we employ to induce the constraints, may yield an arbitrary propositional lineage formula when grounded. This is formally shown in Proposition 2. We note that restrictions on the shape of the constraints, i.e., to avoid the $\#\mathcal{P}$-hard instances of Observation 11, should follow work on tractable probability computations in probabilistic databases. The reason is that the computational complexity arises from the probability computations. In contrast, when solving constraints over deterministic databases, the complexity mainly results from finding a single consistent subset of the database, rather than from counting all of these subsets.

## 4 Temporal-Probabilistic Databases

In recent years, both temporal and probabilistic databases have emerged as two intensively studied areas of database research. So far, the two fields have however been investigated largely only in isolation. In this section, we describe a *closed* and *complete* temporal-probabilistic database (TPDB) model [23], which provides the expressiveness of the afore defined probabilistic database model, but augments this model with temporal annotations for tuples and temporal predicates for the rules. To the best of our knowledge, prior to [23], only Sarma et al. [57] have explicitly modeled time in PDBs. However in the former work time refers to the "transaction-time" of a tuple insertion or update, thus focusing on versioning a probabilistic database. Rather, we consider time as the actual temporal validity of a tuple in the real world (e.g., the time interval of a marriage in the IE scenario).

*Example 18.* This time, our running example is centered around the actors "Robert De Niro" and "Jane Abott" about whom the TPDB of Figure 2 captures a number of facts. Tuple $I_1$ expresses that *DeNiro* was born in *Greenwich* (New York) on August 17th, 1943, which is encoded into the time interval [1943-08-17, 1943-08-18) using an ISO style date/time format. The time and probability

BornIn

| | Subject | Object | Valid Time | $p$ |
|---|---|---|---|---|
| $I_1$ | *DeNiro* | *Greenwich* | [1943-08-17, 1943-08-18) | 0.9 |
| $I_2$ | *DeNiro* | *Tribeca* | [1998-01-01, 1999-01-01) | 0.6 |

Wedding

| | Subject | Object | Valid Time | $p$ |
|---|---|---|---|---|
| $I_3$ | *DeNiro* | *Abbott* | [1936-11-01, 1936-12-01) | 0.3 |
| $I_4$ | *DeNiro* | *Abbott* | [1976-07-29, 1976-07-30) | 0.8 |

Divorce

| | Subject | Object | Valid Time | $p$ |
|---|---|---|---|---|
| $I_5$ | *DeNiro* | *Abbott* | [1988-09-01, 1988-12-01) | 0.8 |

**Fig. 2.** Example Temporal-Probabilistic Database with Tuple Timestamping

annotations together express that this tuple is *true* for the given time interval with probability 0.9, and it is *false* (i.e., it does not exist in the database) for this interval with probability 0.1. Furthermore, tuples are always *false* outside their attached time intervals. Notice that another tuple, $I_2$, states that *DeNiro* could have also been born in *Tribeca* in the interval [1998-01-01, 1999-01-01) with probability 0.6. In the remainders of this section, we investigate how to evaluate queries over this kind of data, i.e., how to propagate time and probabilities from the database to the query answers. We also discuss consistency constraints. For instance, the two tuples of *BornIn* state different birth places of *DeNiro* and create an inconsistency we should rule out by the use of constraints.

### 4.1 Time

We start with the most important point, namely our model of time. As in a calendar, there are a number of choices to make. First, we have to decide on the granularity of time, which could be days, hours or minutes, for instance. Also, we should determine whether time is finite, and, if so, when it starts or ends, e.g., at the first or the last day of a year, respectively.

Technically, we adopt the view of time as points which then can be coalesced to form intervals [37,69]. We consider the *time universe* $\mathcal{U}^T$ as a linearly ordered finite sequence of *time points*, e.g., days, minutes or even milliseconds. Considering time to be finite and discrete later ensures that there are finitely many possible worlds. A *time interval* consists of a contiguous and finite set of ordered time points over $\mathcal{U}^T$, which we denote by a half-open interval $[t_b, t_e)$ where $t_b, t_e \in \mathcal{U}^T$ and $t_b < t_e$. For instance, a day can be viewed as an interval of hours. Moreover, we employ the two constants $t_{min}, t_{max}$ to denote the earliest and latest time point in $\mathcal{U}^T$, respectively. Finally, temporal variables are written as $T$ or $[T_b, T_e)$, if we refer to a time interval.

At this point, we want to remark that we do not consider the finiteness of $\mathcal{U}^T$ to be any limitation of the above model for time in practice, since we can

always choose $t_{min}, t_{max}$ as the earliest and latest time points we observe among the tuples and deduction rules. Also, discrete time points of fixed granularity do not present any restraint, as we could resort to employing time points of smaller granularity than the ones observed in the input data if needed. The complexity of the following operations, which we define over this kind of temporally and probabilistically annotated tuples, will in fact be independent of the granularity of the underlying time universe $\mathcal{U}^T$.

*Example 19.* Regarding the database of Figure 2, $\mathcal{U}^T$ comprises the sequence of days starting at $t_{min}$ := 1936-11-01 and ending at $t_{max}$ := 1999-01-01. We could equally choose any more fine-grained unit for the time points, but for presentation purposes, we select days.

### 4.2 Temporal Relations and Tuples

We now relate data to time, that is, tuples are considered to be valid during a specific time interval, only, and they are invalid outside their attached time intervals. For this, we extend the relations introduced in Section 2.1 to temporal relations, following work by [1,37,66]. We annotate each tuple by a time interval specifying the validity of the tuple over time—a technique, which is commonly referred to as *tuple timestamping* [37]. More specifically, a *temporal relation* $R^T$ is a logical predicate of arity $r \geq 3$, whose latter two arguments are temporal. Hence, an instance of a temporal relation is a finite subset $\mathcal{R}^T \subseteq \mathcal{U}^{r-2} \times \mathcal{U}^T \times \mathcal{U}^T$. Therein, we interpret the temporal arguments $t_b, t_e$ of a tuple $R^T(\bar{a}, t_b, t_e)$ to form a time interval $[t_b, t_e)$. Choosing intervals over time points has the advantage that the storage costs are independent of the granularity of the time points.

*Example 20.* The tuple *BornIn*(*DeNiro*, *Greenwich*, 1943-08-17, 1943-08-18) is valid only at one day, namely on August 17th, 1943.

In general, a temporal relation instance can contain several tuples with equivalent non-temporal arguments $\bar{a}$, but with varying temporal arguments. For instance, assume we have two tuples describing *DeNiro*'s birthday, one time-stamped with the year 1943 and one by the day 1943-08-18. Then, a database engine might conclude that he was born twice on August 18th, 1943 with different probabilities. To resolve this issue, we enforce the time intervals of tuples with identical, non-temporal arguments to be disjoint. A relation instance that adheres to this condition is going to be termed *duplicate-free* [21].

**Definition 16.** *A temporal relation instance $\mathcal{R}^T$ is called* duplicate-free*, if for all pairs of tuples $R^T(\bar{a}, t_b, t_e), R^T(\bar{a}', t'_b, t'_e) \in \mathcal{R}^T$ it holds that:*

$$\bar{a} = \bar{a}' \quad \Rightarrow \quad [t_b, t_e) \cap [t'_b, t'_e) = \emptyset$$

We remark that the above definition does not affect tuples of different, non-temporal arguments or with non-overlapping temporal arguments.

*Example 21.* In Figure 2 the temporal relation instance *Wedding* is duplicate-free, as both tuples have equivalent non-temporal arguments, but their time intervals are non-overlapping.

### 4.3 Temporal-Probabilistic Database Model

In this section, we extend tuple-independent probabilistic databases of Definition 10 to temporal data as in [23]. Intuitively, each tuple has two annotations: a temporal and a probabilistic one. Hence, each tuple exists only during a given time and with a given probability. Supporting both probability and time annotations allows to represent data, where we are unsure whether a tuple is valid at a given set of time points or during an entire time interval, respectively.

**Definition 17.** *For temporal relations $R_1^T, \ldots, R_n^T$ a* tuple-independent temporal-probabilistic database *(TPDB) $(\mathcal{T}, p, \mathcal{U}^T)$ is a triple, where*

1. *$\mathcal{T} := \mathcal{R}_1^T \cup \cdots \cup \mathcal{R}_n^T$ is a finite set of tuples;*
2. *$\forall i \in \{1, \ldots, n\} : \mathcal{R}_i^T$ is duplicate-free;*
3. *all tuples $R_i^T(\bar{a}, t_b, t_e)$ of all relation instances $\mathcal{R}_i^T$ share the time universe $\mathcal{U}^T$, that is, $t_b, t_e \in \mathcal{U}^T$;*
4. *$p$ is a function $p : \mathcal{T} \to (0, 1]$ which assigns a non-zero* probability value *$p(I)$ to each tuple $I \in \mathcal{T}$;*
5. *the probability values of all tuples in $\mathcal{T}$ are assumed to be* independent*.*

In the above definition, the first, fourth and fifth condition are analogous to Definition 10. Still, we here consider *temporal* relation instances $\mathcal{R}_i^T$ and require them to be duplicate-free (see Definition 16). Additionally, all time points occurring in any relation instance $\mathcal{R}_i^T$ must be contained in the time universe $\mathcal{U}^T$. We highlight that the probabilities of two tuples $R(\bar{a}, t_b, t_e)$ and $R(\bar{a}, t'_b, t'_e)$, even if they share $\bar{a}$, are independent due to the fifth condition. In the remaining parts of this chapter, we will thus again drop the attribute "tuple-independent" when we refer to a TPDB. As in Section 3, dependencies among tuples will be induced by constraints and queries.

*Example 22.* The temporal relation instances of Figure 2, together with their time universe defined in Example 19 form the TPDB $(\{I_1, I_2, I_3, I_4, I_5\}, p, \langle$1936-11-01, $\ldots$, 1999-01-01$\rangle)$.

Since tuple probabilities here are defined as in PDBs, and $\mathcal{U}^T$ is finite as well as discrete, the possible-worlds semantics of Subsection 3.1 applies to TPDBs as well. Next, we thus more formally characterize the relationship between TPDBs and (non-temporal) PDBs.

**Proposition 4.** *Every PDB instance $(\mathcal{T}, p)$ can be encoded in a TPDB instance $(\mathcal{T}', p, \mathcal{U}^T)$.*

*Proof.* To achieve the encoding, we create the time universe $\mathcal{U}^T := \langle 1, 2 \rangle$, expand each relation $R$ in $\mathcal{T}$ by two temporal arguments, and set $\mathcal{T}' := \{R^T(\bar{a}, 1, 2) \mid R(\bar{a}) \in \mathcal{T}\}$.

### 4.4 Temporal Arithmetic Predicates

To express temporal statements, it is necessary to be able to compare temporal annotations in the form of time points. Hence, we support two temporal-arithmetic predicates "$=^T$" and "$<^T$" [29,46], which each check for the equality and precedence of two time points, respectively.

**Definition 18.** *For $t_1, t_2 \in \mathcal{U}^T$ the* temporal-arithmetic predicates $=^T$ *and* $<^T$ *are evaluated as follows:*

$$t_1 =^T t_2 \equiv \begin{cases} \text{true } \textit{if } t_1 = t_2, \\ \text{false } \textit{otherwise}, \end{cases}$$
$$t_1 <^T t_2 \equiv \begin{cases} \text{true } \textit{if } t_1 \textit{ strictly before } t_2 \textit{ in } \mathcal{U}^T, \\ \text{false } \textit{otherwise}. \end{cases}$$

In other words, "$=^T$" is satisfied, whenever two time points are identical, whereas "$<^T$" compares the order of two time points in $\mathcal{U}^T$.

*Example 23.* Since 1998-01-01 is before 1999-01-01, we have 1998-01-01 $<^T$ 1999-01-01 $\equiv$ *true*.

By utilizing conjunctions of "$<^T$" and "$=^T$" predicates over the temporal arguments, we are able to express all of the 13 relationships between time intervals defined in the seminal work of Allen [4], such as *overlaps*, *disjoint* or *starts*.

**Proposition 5.** *We can express all the 13 relationships between two time intervals as defined by Allen [4] by relying solely on conjunctions of "$=^T$" and "$<^T$".*

*Proof.*

| Allen's Relation | Encoding |
|---|---|
| $[T_b, T_e)$ before $[T'_b, T'_e)$ | $T_e <^T T'_b$ |
| $[T_b, T_e)$ equal $[T'_b, T'_e)$ | $T_b =^T T'_b \wedge T_e =^T T'_e$ |
| $[T_b, T_e)$ meets $[T'_b, T'_e)$ | $T_e =^T T'_b$ |
| $[T_b, T_e)$ overlaps $[T'_b, T'_e)$ | $T_b <^T T'_b \wedge T'_b <^T T_e \wedge T_e <^T T'_e$ |
| $[T_b, T_e)$ during $[T'_b, T'_e)$ | $T'_b <^T T_b \wedge T_e <^T T'_e$ |
| $[T_b, T_e)$ starts $[T'_b, T'_e)$ | $T_b =^T T'_b \wedge T_e <^T T'_e$ |
| $[T_b, T_e)$ finishes $[T'_b, T'_e)$ | $T'_b <^T T_b \wedge T_e =^T T'_e$ |

The remaining 6 relationships are the inverse of one of the above ones, except for equality which is symmetric.

### 4.5 Temporal Deduction Rules

Next, we devise temporal deduction rules, that is, general "if-then" rules which mention time. Formally, our temporal deduction rules [23] are logical implications over temporal relations and temporal arithmetic predicates, defined as follows.

**Definition 19.** *A* temporal deduction rule *is a logical rule of the form*

$$R^T(\bar{X}, T_b, T_e) \leftarrow \bigwedge_{i=1,\ldots,n} R_i^T(\bar{X}_i, T_{i,b}, T_{i,e}) \ \wedge \bigwedge_{j=1,\ldots,m} \neg R_j^T(\bar{X}_j, T_{j,b}, T_{j,e}) \ \wedge \Phi(\bar{X}_A, \bar{T}_A) \tag{10}$$

*where*

1. *all requirements of Definition 2 hold;*
2. *$T_b$, $T_e$, $T_{i,b}$, $T_{i,e}$, $T_{j,b}$, $T_{j,e}$ and $\bar{T}_A$ are temporal constants and variables, where $Var(T_b, T_e)$, $Var(T_{j,b}, T_{j,e})$, $Var(\bar{T}_A) \subseteq \bigcup_i Var(T_{i,b}, T_{i,e})$;*
3. *$\Phi(\bar{X}_A, \bar{T}_A)$ is a conjunction of literals over the arithmetic predicates, such as "$=$" and "$\neq$", and the temporal arithmetic predicates "$=^T$" and "$<^T$".*

With respect to non-temporal arguments, all restrictions of non-temporal deduction rules (see Definition 2) hold. Combining this observation with the second requirement above, we conclude that temporal deduction rules are *safe* [2]. Furthermore, the third condition allows the temporal-arithmetic predicates of Definition 18 to occur in temporal deduction rules. Of course, also non-temporal relations are allowed in temporal deduction rules, hence inducing mixtures of temporal and non-temporal rules. We note that the above class of temporal deduction rules is very expressive, as it allows $T_b$, $T_e$ to be constants or to be variables from different literals $R_i^T$. As before, we assume also the temporal deduction rules to be *non-recursive*.

*Example 24.* Given the tuples of Figure 2 about both *DeNiro*'s wedding and divorce with *Abbott*, we aim to deduce the time interval of their marriage by temporal deduction rules. The first rule states that a couple stays married from the begin time point of their wedding (denoted by the variable $T_{b,1}$) until the last possible time point we consider (denoted by the constant $t_{max}$), unless there is a divorce tuple.

$$Marriage^T(P_1, P_2, T_{b,1}, t_{max}) \leftarrow \left( \begin{array}{c} Wedding^T(P_1, P_2, T_{b,1}, T_{e,1}) \ \wedge \\ \neg Divorce(P_1, P_2) \end{array} \right) \tag{11}$$

Here, the existence of a divorce independent of time is modeled by the following projection:

$$Divorce(P_1, P_2) \leftarrow Divorce^T(P_1, P_2, T_b, T_e)$$

The second rule states that a couple stays married from the begin time point of their wedding till the end time point of their divorce.

$$Marriage^T(P_1, P_2, T_{b,1}, T_{e,2}) \leftarrow \left( \begin{array}{c} Wedding^T(P_1, P_2, T_{b,1}, T_{e,1}) \ \wedge \\ Divorce^T(P_1, P_2, T_{b,2}, T_{e,2}) \ \wedge \\ T_{e,1} <^T T_{b,2} \end{array} \right) \tag{12}$$

Thereby, we consider only weddings that took place before divorces as stated by the condition $T_{e,1} <^T T_{b,2}$.

### 4.6 Lineage and Deduplication

As in Section 3.4, we trace the deduction history of tuples via lineage, however, with the additional twist that lineage may now also vary over time. Since temporal deduction rules are safe, the groundings $G(D, \mathcal{T})$ of Definition 4 and the new tuples $\mathit{IntensionalTuples}(D, \mathcal{T})$ of Definition 5 apply to temporal deduction rules as well. Hence, at first glance lineage tracing according to Definition 11 works in a temporal context, but with one random variable for each tuple with its time interval. However, if we execute temporal deduction rules, the newly derived tuples may not necessarily define a duplicate-free relation instance. We illustrate this issue by the following example.

*Example 25.* Let the deduction rules of Example 24 and the tuples of Figure 2 be given. Now, in Figure 3, we visualize both the tuples from database (at the bottom) and the deduced tuples (in the middle). Inspecting the deduced tuples,

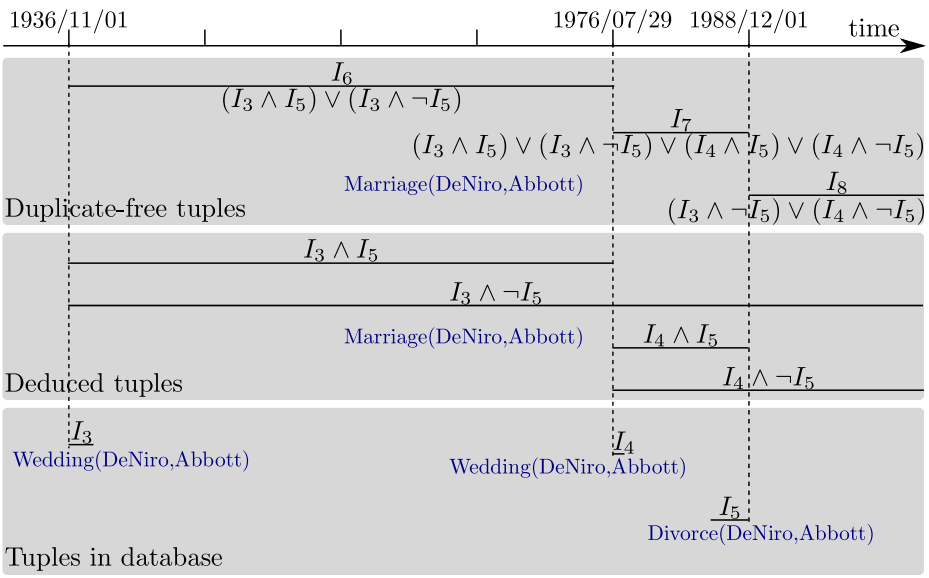



**Fig. 3.** Deducing and Deduplicating Tuples with Time Intervals

we realize that they have equivalent non-temporal arguments, i.e., *DeNiro* and *Abbott*, but their time intervals are overlapping, which contradicts Definition 16 of duplicate-free relation instances.

Hence, in order to convert a temporal relation instance with duplicates (as shown in the middle of Figure 3) into a duplicate-free temporal relation (as shown on the top of Figure 3), we provide the following definition.

**Definition 20.** *Let a temporal relation $R^T$, non-temporal constants $\bar{a}$, a time point $t \in \mathcal{U}^T$, and a set of tuples $\mathcal{T}$ be given. Then, $L$ is defined as the set of lineages of tuples $R^T(\bar{a}, t_b, t_e)$ that are valid at time point $t$:*

$$L(R^T, \bar{a}, t, \mathcal{T}) := \{\lambda(I) \mid I = R^T(\bar{a}, t_b, t_e) \in \mathcal{T}, t_b \leq t < t_e\}$$

*We create* duplicate free tuples $I' = R^T(\bar{a}, t_b, t_e)$ *such that for any pair of time points $t_0, t_1 \in [t_b, t_e)$ it holds that:*

$$L(R^T, \bar{a}, t_0, \mathcal{T}) = L(R^T, \bar{a}, t_1, \mathcal{T}) \tag{13}$$

*Furthermore, we define the new tuples' lineage to be:*

$$\lambda(I') := \bigvee_{\phi_i \in L(R^T, \bar{a}, t_b, \mathcal{T})} \phi_i \tag{14}$$

In short, for each time point $t$, we create the disjunction of all tuples being valid at $t$ (see Equation 14). More detailed, for a given relation instance and the non-temporal arguments of a tuple, $L$ is the set of all tuples' lineages that share the same non-temporal arguments and which are valid at time point $t$. Hence, consecutive time points for which $L$ contains the same lineage formulas form the new intervals (see Equation (13)).

We remark that for the equality of Equation (13), we focus on syntactical equivalence checks between the lineage formulas. We thus refrain from full (i.e., logical) equivalence checks, as they are known to be *co-*$\mathcal{NP}$-complete [11].

*Example 26.* Applying Definition 20 to the tuples in the middle of Figure 3 yields the tuples shown at the top of the figure. For instance, if we inspect $L$ at the time points 1976-07-28 and 1976-07-29, we notice that $\{I_3 \wedge I_5, I_3 \wedge \neg I_5\} \neq \{I_3 \wedge I_5, I_3 \wedge \neg I_5, I_4 \wedge I_5, I_4 \wedge \neg I_5\}$, so two different result tuples $I_6$ and $I_7$ have to be kept in the relation. In total, the resulting duplicate-free tuples are:

Marriage

| | Subject | Object | Valid Time |
|---|---|---|---|
| $I_6$ | *DeNiro* | *Abbott* | $[$1936-11-01, 1976-07-29$)$ |
| $I_7$ | *DeNiro* | *Abbott* | $[$1976-07-29, 1988-12-01$)$ |
| $I_8$ | *DeNiro* | *Abbott* | $[$1988-12-01, $t_{max})$ |

Following Equation (14), their respective lineages are:

$$\lambda(I_6) = (I_3 \wedge I_5) \vee (I_3 \wedge \neg I_5)$$
$$\lambda(I_7) = (I_3 \wedge I_5) \vee (I_3 \wedge \neg I_5) \vee (I_4 \wedge I_5) \vee (I_4 \wedge \neg I_5)$$
$$\lambda(I_8) = (I_3 \wedge \neg I_5) \vee (I_4 \wedge \neg I_5)$$

Hence, for temporal deduction rules the combination of Definitions 11 and 20 creates the lineage formulas. We want to remark that these lineage formulas are guaranteed to yield purely propositional formulas, as it is captured by the following observation.

**Observation 12** *Temporal deduction rules and temporal deduplication produce propositional lineage formulas without any explicit mentioning of time.*

Hence, any work on PDBs with lineage can be applied to TPDBs as well, especially also works on efficient probability computations (see Section 3.5).

### 4.7 Queries and Query Answers

As a final step, we introduce temporal queries which extend Definition 6 by a temporal component. Thus, in analogy to the atemporal case, a temporal query again resembles the body of a temporal deduction rule.

**Definition 21.** *Given temporal deduction rules $\mathcal{D}$ with their intensional relations a* temporal query *$Q$ is a conjunction:*

$$Q(\bar{X},\bar{T}) \;:=\; \bigwedge_{i=1,\ldots,n} R_i^T(\bar{X}_i, T_{i,b}, T_{i,e}) \;\wedge\; \bigwedge_{j=1,\ldots,m} \neg R_j^T(\bar{X}_j, T_{j,b}, T_{j,e}) \wedge \Phi(\bar{X}_A, \bar{T}_A)$$

*where*

1. *all requirements of Definition 6 hold;*
2. *$\bar{T}$, $T_{i,b}$, $T_{i,e}$, $T_{j,b}$, $T_{j,e}$ and $\bar{T}_A$ are temporal constants and variables, which satisfy:*
   (a) *$Var(\bar{T}) = \bigcup_{i=1,\ldots,n} Var(T_{i,b}, T_{i,e})$;*
   (b) *$Var(\bar{T}_A) \subseteq \bigcup_{i=1,\ldots,n} Var(T_{i,b}, T_{i,e})$;*
   (c) *for all $j \in \{1,\ldots,m\}$ it holds that $Var(T_{j,b}, T_{j,e}) \subseteq \bigcup_{i=1,\ldots,n} Var(T_{i,b}, T_{i,e})$;*
3. *$Var(\bar{X}) \cup Var(\bar{T})$ denote the* query variables*;*
4. *$\Phi(\bar{X}_A, \bar{T}_A)$ is a conjunction of (temporal) arithmetic literals.*

Temporal queries thus inherit all properties from their non-temporal counterparts (see Definition 6), in particular that all relations occurring in the query are intensional. The first and second condition above ensure *safe queries* [2]. In this section, the query variables are formed by both the variables in $\bar{X}$ and in $\bar{T}$. With respect to arithmetic predicates, we support both non-temporal ones as in Section 2.4 and additionally the temporal ones from Definition 18.

*Example 27.* If we are interested in people who were married before 1980, we write the query

$$Marriage(P_1, P_2, T_b, T_e) \;\wedge\; T_b <^T \text{1980-01-01}$$

where the intensional relation *Marriage* is defined as in Example 24.

Since the restrictions on variables of Definition 6 and Definition 21 coincide, query answers can be obtained as in the non-temporal case of Definition 7.

### 4.8 Temporal Consistency Constraints

In Subsection 3.6, we introduced constraints as propositional lineage formula $\phi_c$. Following Definition 14, we can create constraints via deduction rules. For this, we keep two sets of literals $\mathcal{C}_p$ and $\mathcal{C}_n$ which again relate to constraints that must always hold and must never hold, respectively. Then, the literals of both sets induce the lineage formula $\phi_c$ (see Definition 14). Hence, in this section constraints are formulated as temporal deduction rules. As we support temporal-arithmetic predicates (see Definition 18) in the temporal deduction rules, we can express any temporal precedence (i.e., ordering) constraint, and any temporal disjointness or containment constraint.

*Example 28.* If we intend to enforce that persons are born before their marriage starts, we write

$$\mathit{Constraint}(P_1, P_2, T_b, T_e, T'_b, T'_e) \leftarrow \begin{pmatrix} \mathit{Born}^T(P_1, T_b, T_e) \wedge \\ \mathit{Marriage}^T(P_1, P_2, T'_b, T'_e) \wedge \end{pmatrix} T'_b <^T T_e \tag{15}$$

and add $\mathit{Constraint}(P_1, P_2, T_b, T_e, T'_b, T'_e)$ to $\mathcal{C}_n$. To abbreviate this notation, we also write the above constraint as:

$$\neg(\mathit{Born}^T(P_1, T_b, T_e) \wedge \mathit{Marriage}^T(P_1, P_2, T'_b, T'_e) \wedge T'_b <^T T_e)$$

Here, the negation resembles that the head literal of Equation (15) is in $\mathcal{C}_n$, i.e., it should never hold. When we ground the above constraint, all pairs of $\mathit{Marriage}^T$ and $\mathit{Born}^T$ tuples contradicting the correct temporal ordering are excluded by $\phi_c$.

### 4.9 Closure and Completeness

Generally, a representation formalism is called *complete* [33] if it can represent any finite instance of data, which in our case is temporal and probabilistic. Furthermore, a representations system is *closed* [33] if all query results can be expressed in the representation itself.

**Theorem 1.** *A TPDB* $(\mathcal{T}, p, \mathcal{U}^T)$ *with lineage is* closed *and* complete *under all relational operations which are expressible by the temporal deduction rules* $\mathcal{D}$*.*

Because completeness is the stronger requirement, which also implies closure, we next provide a proof for the completeness of our TPDB model.

*Proof.* We show that, when given any finite instance $\mathcal{T}$ of temporal and probabilistic relational data, we can represent it in our TPDB model. Without loss of generality, we are given only one relation instance $\mathcal{R}^T$ along with its possible worlds $\mathcal{W}_1, \ldots, \mathcal{W}_n$ and a probability $P(\mathcal{W}_i)$ for each of them. Now, to encode these in a TPDB $(\mathcal{T}, p, \mathcal{U}^T)$, there are three points to show, namely (1) setting $\mathcal{U}^T$, (2) ensuring that $\mathcal{R}^T$ is duplicate free, and (3) determining $\mathcal{T}$ and $p$.

First, we select the earliest and latest time points $t_{min}$ and $t_{max}$, respectively, which occur in $\mathcal{R}^T$. From this, we create the sequence $\mathcal{U}^T := \langle t_{min}, \ldots, t_{max} \rangle$ where each time point is of the smallest granularity of time points that occurs in $\mathcal{R}^T$. Second, to guarantee that each $\mathcal{R}^T$ is duplicate-free (see Definition 16), we create a new relation instance $\mathcal{R}^{T'}$ which extends each tuple by a unique id, e.g., if $R^T(\bar{a}, t_b, t_e) \in \mathcal{R}^T$, then $R^{T'}(id, \bar{a}, t_b, t_e) \in \mathcal{R}^{T'}$. Third, regarding the probabilistic data, we follow [7,65] by proving the statement via induction over the number of possible worlds. Let the possible worlds $\mathcal{W}_i$ range over $\mathcal{R}^{T'}$.

Basis $n = 1$:

In this case, there is only one possible world $\mathcal{W}_1$ with $P(\mathcal{W}_1)$. We store $\mathcal{W}_1$ in the deterministic relation $R_1^{T',d}$ and create an uncertain relation $R^u(X)$ holding exactly one tuple $R^u(1)$ with $p(R^u(1)) = 1$. Then, the rule

$$R_1^{T'}(\bar{X}) \leftarrow R_1^{T',d}(\bar{X}) \wedge R^u(1)$$

along with $\mathcal{T}_1 := \mathcal{W}_1$ encodes the TPDB. Now, queries posed on $R_1^{T'}$ deliver the correct semantics.

Step $n \rightarrow n + 1$:

We want to extend the TPDB by a possible world $\mathcal{W}_{n+1}$ which should have $P(\mathcal{W}_{n+1}) = p_{n+1}$. For this, we create the deterministic relation $R_{n+1}^{T',d}$ containing the tuples of $\mathcal{W}_{n+1}$. Then, we insert the tuple $R^u(n+1)$ into $R^u$ and set its probability value to $p_{n+1}$. Now, we add the rules:

$$R_{n+1}^{T'}(\bar{X}) \leftarrow R_{n+1}^{T',d}(\bar{X}) \wedge R^u(n+1)$$
$$R_{n+1}^{T'}(\bar{X}) \leftarrow R_n^{T'}(\bar{X}) \wedge \neg R^u(n+1)$$

Next, we set $\mathcal{T}_{n+1} := \mathcal{T}_n \cup \mathcal{W}_{n+1}$ to finalize the construction of the resulting TPDB. Again, queries formulated on $R_{n+1}^{T'}$ yield the intended semantics.

## 5 Top-k Query Processing

Motivated by queries whose probability computations entail $\#\mathcal{P}$-hard [16,15] instances, the query evaluation problem in PDBs has been studied very intensively [16,15,17,38,42,53,59]. Except for two works [50,51], which we are aware of, each of these approaches aims for computing all answers along with their probabilities. Still, among these answers many of them may exhibit low probabilities, thus indicating, for example, that we are not very confident in them or that they are unlikely to exist. To avoid this, in our recent work [25], we opt for returning only the *top-k query answers*, ranked by their probabilities. Besides the benefit of presenting only the high-probability answers to the user, top-$k$ approaches allow for significant runtime speed-ups. The reason is that we can save on computations for the neglected low-probability answers. Thus, an algorithmic approach for bounding the probabilities of the top-$k$ answers represented via a novel notion of *first-order lineage* formulas is developed in this section.

*Example 29.* Figure 4 depicts a probabilistic database in the movie domain. By the given deduction rules we intend to derive actors and directors who are known for working on movies in the crime genre as expressed by the query $KnownFor(X, Crime)$.

Query:

$$KnownFor(X, Crime)$$

Deduction Rules:

$$\begin{aligned}
KnownFor(X,Y) &\leftarrow BestDirector(X,Z) \wedge Category(Z,Y)\\
KnownFor(X,Y) &\leftarrow WonAward(Z, BestPicture) \wedge ActedOnly(X,Z) \wedge Category(Z,Y)\\
BestDirector(X,Z) &\leftarrow Director(X,Z) \wedge WonAward(Z, BestDirector)\\
ActedOnly(X,Z) &\leftarrow ActedIn(X,Z) \wedge \neg Directed(X,Z)
\end{aligned}$$

Probabilistic Database Tuples:

Directed

| | Director | Movie | $p$ |
|---|---|---|---|
| $I_1$ | *Coppola* | *ApocalypseNow* | 0.8 |
| $I_2$ | *Coppola* | *Godfather* | 0.9 |
| $I_3$ | *Tarantino* | *PulpFiction* | 0.7 |

ActedIn

| | Actor | Movie | $p$ |
|---|---|---|---|
| $I_4$ | *Brando* | *ApocalypseNow* | 0.6 |
| $I_5$ | *Pacino* | *Godfather* | 0.3 |
| $I_6$ | *Tarantino* | *PulpFiction* | 0.4 |

WonAward

| | Movie | Award | $p$ |
|---|---|---|---|
| $I_7$ | *ApocalypseNow* | *BestScript* | 0.3 |
| $I_8$ | *Godfather* | *BestDirector* | 0.8 |
| $I_9$ | *Godfather* | *BestPicture* | 0.4 |
| $I_{10}$ | *PulpFiction* | *BestPicture* | 0.9 |

Category

| | Movie | Category | $p$ |
|---|---|---|---|
| $I_{11}$ | *ApocalypseNow* | *War* | 0.9 |
| $I_{12}$ | *Godfather* | *Crime* | 0.5 |
| $I_{13}$ | *PulpFiction* | *Crime* | 0.9 |
| $I_{14}$ | *Inception* | *Drama* | 0.6 |

**Fig. 4.** Example PDB with a Query and Deduction Rules

When we execute the query, we obtain the following three answers together with their lineages:

| Answer | Lineage | Probability |
|---|---|---|
| $KnownFor(Coppola, Crime)$ | $I_2 \wedge I_8 \wedge I_{12}$ | 0.36 |
| $KnownFor(Tarantino, Crime)$ | $I_{10} \wedge I_6 \wedge \neg I_3 \wedge I_{13}$ | 0.10 |
| $KnownFor(Pacino, Crime)$ | $I_9 \wedge I_5 \wedge I_{12}$ | 0.06 |

Now, imagine we are not interested in all answers, as in the table above, but rather in the most probable answer, e.g., $KnownFor(Coppola, Crime)$. This is the setting of the present section. We will elaborate on how to compute the $k$ most likely answers efficiently by (1) not fully computing lineage and (2) pruning other less probable answers, such as $KnownFor(Tarantino, Crime)$ and $KnownFor(Pacino, Crime)$, as early as possible.

### 5.1 First-Order Lineage

To handle partial grounding states, we next extend the definition of propositional lineage from Subsection 3.4 to a new notion of first-order lineage [25], which hence can contain variables and quantifiers. In contrast to propositional lineage, a first-order lineage formula does not necessarily represent a single query answer, but may rather represent entire sets of answers via variables that have not been bound to constants by the grounding procedure yet. Each distinct query answer in such a set will thus be characterized by constants once the query variables become bound by the grounding procedure.

Throughout this section, we assume the extensional relations to be duplicate-free. That is, there is no pair of tuples having the same arguments. This assumption facilitates the theoretical analysis which follows. Still, in practice, we could always remove potential duplicates by an *independent-or* projection over the input relations as a preprocessing step.

**Deduction Rules with Quantifiers.** To facilitate the construction of first-order lineage, we will write out the existential quantifiers that occur only in the bodies of the deduction rules explicitly, which is captured more precisely by the following definition.

**Definition 22.** *A* first-order deduction rule *is a logical rule of the form*

$$R(\bar{X}) \leftarrow \exists \bar{X}_e \bigwedge_{i=1,\ldots,n} R_i(\bar{X}_i) \wedge \bigwedge_{j=1,\ldots,m} \neg R_j(\bar{X}_j) \quad \wedge \Phi(\bar{X}_A)$$

*where*

1. *all requirements of Definition 2 hold;*
2. $\bar{X}_e = (\bigcup_{i=1,\ldots,n} Var(\bar{X}_i)) \setminus Var(\bar{X})$

The difference to Definition 2 might seem subtle, but we this time explicitly enforce all variables $\bar{X}_e$, which occur in positive literals $R_i(\bar{X}_i)$, but not in the head $R(\bar{X})$, to be existentially quantified. This still is in accordance to standard Datalog semantics [2]. Later however, when constructing first-order lineage formulas, we will need to trace and maintain the existential quantifiers in the lineage formulas explicitly.

*Example 30.* Let us adapt the deduction rules of Figure 4 to Definition 22 by writing the quantifiers explicitly:

$$\begin{aligned}
KnownFor(X,Y) &\leftarrow \exists Z\ BestDirector(X,Z) \wedge Category(Z,Y) \\
KnownFor(X,Y) &\leftarrow \exists Z \left( \begin{array}{c} WonAward(Z, BestPicture) \wedge ActedOnly(X,Z) \\ \wedge\ Category(Z,Y) \end{array} \right) \\
BestDirector(X,Z) &\leftarrow Director(X,Z) \wedge WonAward(Z, BestDirector) \\
ActedOnly(X,Z) &\leftarrow ActedIn(X,Z) \wedge \neg Directed(X,Z)
\end{aligned}$$

**Top-Down Grounding with First-Order Lineage.** Our main observation for this section is that first-order lineage can be constructed from a top-down grounding procedure in Datalog and can thus capture any intermediate state in this process. In a top-down approach, we start at the query literals and iteratively expand the deduction rules until we reach the database tuples. In Section 3.4, the direction was reversed, since we started at the database until we ended up at the query. As first theoretical tool, we establish consistent vectors of constants $\bar{a}$ and mixtures of variables and constants $\bar{X}$. This technique enables us to match first-order literals against database tuples.

**Definition 23.** *Let $X_i$ and $a_i$ denote the $i$-th entry in the vector of variables and constants $\bar{X}$ and the vector of constants $\bar{a}$, respectively. We call $\bar{X}$ and $\bar{a}$* consistent*, if*

$$\forall X_i \in \bar{X} : X_i \text{ is a constant} \Rightarrow X_i = a_i$$

In other words, all constants in the vector $\bar{X}$ have to match the constant in $\bar{a}$ at the respective position.

*Example 31.* The vectors (*X*, *Crime*) and (*Coppola*, *Crime*) are consistent, as the constant in the second entry occurs in both vectors.

Based on consistent vectors, we gather all constants binding a variable in a set of tuples. Later, this allows us to collect all tuples from the database, which match a first-order literal.

**Definition 24.** *Let $\mathcal{T}$ be a set of tuples and $R(\bar{X})$ be a literal with extensional relation $R$. Then, the set of constants from $\mathcal{T}$, which* bind *the variable $X_i$ in $\bar{X}$ is:*

$$\mathit{Bindings}(X_i, R(\bar{X}), \mathcal{T}) := \{a_i \mid R(\bar{a}) \in \mathcal{T}, \bar{X} \text{ and } \bar{a} \text{ consistent}\}$$

We note that $a_i$ and $X_i$ refer to $i$-th entry of $\bar{a}$ and $\bar{X}$, respectively. In general, the above set can be empty or reach the same cardinality as $\mathcal{T}$.

*Example 32.* Let the tuples of Figure 4 establish $\mathcal{T}$. Then, considering the literal *Directed*(*Coppola*, *Y*) we obtain the following bindings for the variable *Y*:

$$\mathit{Bindings}(Y, \mathit{Directed}(\mathit{Coppola}, Y), \mathcal{T}) = \{\mathit{ApocalypseNow}, \mathit{Godfather}\}$$

The last technical prerequisite before introducing the construction of first-order lineage are logical equivalences which eliminate quantifiers. For this, assuming that $a_1, \ldots, a_n$ are all possible constants for the variable $X$, then the following two equivalences [2,67] hold:

$$\begin{aligned} \exists X \Phi &\equiv \sigma_{a_1}(\Phi) \vee \cdots \vee \sigma_{a_n}(\Phi) \\ \forall X \Phi &\equiv \sigma_{a_1}(\Phi) \wedge \cdots \wedge \sigma_{a_n}(\Phi) \end{aligned} \tag{16}$$

Here, $\sigma_{a_i}$ is shorthand for $\sigma(X) = a_i$. Finally, we establish the top-down counterpart to Definition 11 for first-order lineage. We create first-order lineage by starting with the query literals and then by iteratively replacing the first-order literals by the bodies of the respective deduction rules and, finally, by tuples from the database.

**Definition 25.** *Let a set of tuples $\mathcal{T}$, a set of deduction rules $\mathcal{D}$, a first-order lineage formula $\Phi$, and a literal $R(\bar{X})$ which occurs in $\Phi$ be given. We define the* expansion *of $R(\bar{X})$ in $\Phi$ by a function:*

$$SLD : Literals \times FirstOrderLineage \to Set[FirstOrderLineage]$$

*In detail:*

1. *If $R$ is* intensional*, then:*

$$SLD(R(\bar{X}), \Phi) := \left\{ \Phi[R(\bar{X}) / \bigvee_{(R(\bar{X}') \leftarrow \Psi) \in \mathcal{D}} \sigma_{\bar{X}}(\Psi)] \right\}$$

   *where $\sigma_{\bar{X}}$'s image coincides with $\bar{X}$.*
2. *If $R$ is* extensional*, we initialize:*

$$S_0 := \{\Phi\}$$

   *and then iterate over all variables $X \in Var(\bar{X})$:*
   *(a) If $X$ is a query variable:*

$$S_i := \{\sigma_a(\Phi') \mid \Phi' \in S_{i-1}, a \in Bindings(X, R(\bar{X}), \mathcal{T})\}$$

   *where $\sigma_a(X) = a$.*
   *(b) If $X$ is bound by $\exists X$, then we replace the subformula $\exists X\ \Psi$ of $\Phi$ in $S_i$ by $\sigma_{a_1}(\Psi) \vee \cdots \vee \sigma_{a_n}(\Psi)$ where all $a_i \in Bindings(X, R(\bar{X}), \mathcal{T})$.*
   *(c) If $X$ is bound by $\forall X$, then we replace the subformula $\forall X\ \Psi$ of $\Phi$ in $S_i$ by $\sigma_{a_1}(\Psi) \wedge \cdots \wedge \sigma_{a_n}(\Psi)$ where all $a_i \in Bindings(X, R(\bar{X}), \mathcal{T})$.*
   *Finally, we replace all ground literals $R(\bar{a})$ in the last $S_i$ by their tuple identifier $I$ and assign $SLD(R(\bar{X}), \Phi) := S_i$.*
3. *If there is* no match *to $R(\bar{X})$, neither in $\mathcal{T}$ nor in $\mathcal{D}$, then:*

$$SLD(R(\bar{X}), \Phi) := \{\Phi_{[R(\bar{X})/false]}\}$$

4. *If $R$ is* arithmetic *and $Var(\bar{X}) = \emptyset$, then we evaluate $R(\bar{X})$ to a constant truth value $V$ (thus assigning* true *or* false*), and we set:*

$$SLD(R(\bar{X}), \Phi) := \{\Phi_{[R(\bar{X})/V]}\}$$

The above definition is admittedly more involved than the previous definition of propositional lineage. In the first case, we address *intensional literals* $R(\bar{X})$, where we exchange $R(\bar{X})$ for the disjunction of the deduction rules having $R$ in their head literal. Since $\bar{X}$ can contain constants, we propagate them to the rules' bodies by writing $\sigma_{\bar{X}}(\Psi)$. *Extensional literals*, which are the subject of the second case, can yield sets of first-order lineage formulas. We proceed by considering each variable individually and distinguish between query variables (see Definition 6), existentially bound variables, and universally bound variables. If $X$ is a *query variable*, then each constant $a$ that yields a valid binding to $X$ produces a new distinct set of query answers represented by the lineage formula $\sigma_a(\Phi')$.

Conversely, if $X$ is *existentially quantified*, we apply Equation (16) to expand the formula by introducing a disjunction ranging over the constants $a_1, \ldots, a_n$ which bind $X$. Analogously, a *universally quantified* $X$ yields the conjunction over the constants $a_1, \ldots, a_n$. The third case again reflects a closed-world assumption [2], where we replace a literal with *no match* by the constant *false*. Finally, if we have an *arithmetic* literal that has only constants as arguments, we evaluate it to its truth value. We can safely assume that all arguments of the arithmetic literal are finally going to be bound to constants, because these must be bound to at least one positive, relational literal (see Definition 2). What we omitted for brevity are constants in the head literal of a deduction rule. Since these constants bind variables as in extensional literals (the second case), a mixture of the first and second case arises.

*Example 33.* We illustrate Definition 25 by providing an example for each case. As for $\mathcal{T}$ we assume it to comprise all tuples of Figure 4.

1. We expand the formula $\Phi := \mathit{KnownFor}(X, \mathit{Crime})$ over the deduction rules of Example 30. Since *KnownFor* is an intensional relation, we start with the first case of Definition 25. There, the substitution $\sigma_{\bar{X}}$ binds the second argument to *Crime*:

   $$\sigma_{\bar{X}}(Y) = \mathit{Crime}$$

   Since there are two rules having *KnownFor* in the head literal we apply the substitution to both bodies which then yields:

   $$\left\{ \begin{array}{c} (\exists Z\ \mathit{BestDirector}(X, Z) \wedge \mathit{Category}(Z, \mathit{Crime})) \\ \vee \\ \left( \exists Z \begin{array}{c} \mathit{WonAward}(Z, \mathit{BestPicture}) \wedge \\ \mathit{ActedOnly}(X, Z) \wedge \mathit{Category}(Z, \mathit{Crime}) \end{array} \right) \end{array} \right\}$$

2. (a) Imagine we are given the first-order lineage formula

   $$\Phi := \mathit{BestDirector}(X, Z) \wedge \mathit{Category}(Z, \mathit{Crime})$$

   and we intend to expand the literal $\mathit{Category}(Z, \mathit{Crime})$. Here, *Category* is an extensional relation. First, we determine the bindings of $Z$, which are *Godfather* and *PulpFiction*. Since $Z$ is not quantified, but a query variable, we obtain several formulas, one for each of the constants:

   $$\left\{ \begin{array}{l} (\mathit{BestDirector}(X, \mathit{Godfather}) \wedge \mathit{Category}(\mathit{Godfather}, \mathit{Crime})), \\ (\mathit{BestDirector}(X, \mathit{PulpFiction}) \wedge \mathit{Category}(\mathit{PulpFiction}, \mathit{Crime})) \end{array} \right\}$$

   (b) In this case, we quantify $Z$ existentially and otherwise keep the previous formula:

   $$\Phi := \exists Z\ \mathit{BestDirector}(X, Z) \wedge \mathit{Category}(Z, \mathit{Crime})$$

Then, we expand the *Category* literal by case 2(b) of Definition 25 which results in a disjunction over the two constants *Godfather* and *PulpFiction*:

$$\left\{ \begin{array}{c} (BestDirector(X, Godfather) \wedge Category(Godfather, Crime)) \\ \vee \\ (BestDirector(X, PulpFiction) \wedge Category(PulpFiction, Crime)) \end{array} \right\}$$

(c) Let us consider a universal quantifier instead:

$$\Phi := \forall Z \ BestDirector(X, Z) \wedge Category(Z, Crime)$$

When applying a SLD step to the *Category* literal, we instantiate $Z$ by the two constants *Godfather* and *PulpFiction* to obtain the conjunction:

$$\left\{ \begin{array}{c} (BestDirector(X, Godfather) \wedge Category(Godfather, Crime)) \\ \wedge \\ (BestDirector(X, PulpFiction) \wedge Category(PulpFiction, Crime)) \end{array} \right\}$$

3. Trying to resolve the second literal of

$$\Phi := \exists Z \ BestDirector(X, Z) \wedge Category(Z, Comedy)$$

over $\mathcal{T}$ delivers no result. Hence, we replace it by *false* which yields:

$$\{\exists Z \ BestDirector(X, Z) \wedge false\}$$

4. In the last case, we have an arithmetic literal, for example

$$I_1 \wedge I_2 \wedge ApocalypseNow \neq Godfather$$

which we then evaluate to $I_1 \wedge I_2 \wedge true$.

Analogously to the Disjunctive Normal Form (DNF) for propositional formulas, any first-order formula can equivalently be transformed into prenex form by pulling all quantifiers in front of the formula. The remaining formula can again be transformed into DNF, which is then called Prenex Disjunctive Normal Form (PDNF) [67].

Next, we devise two formal properties of first-order lineage formulas. First, the existence of at least one proof implies that all query variables are bound. Second, unbound query variables imply that a first-order lineage formula represents a set of query answers.

**Proposition 6.** *Expanding a query $Q(\bar{X})$ with query variables $\bar{X}$ to first-order lineage by repeatedly applying Definition 25 has the following properties:*

1. *If at least one clause in the disjunctive normal form of the lineage formula is propositional, then all query variables $\bar{X}$ are bound to constants.*
2. *If at least one query variable $X \in \bar{X}$ is unbound a lineage formula represents a (potentially empty) set of query answers.*

*Proof.* We prove both statements separately.

1. Without loss of generality, we assume the formula to be in PDNF. Then, every clause stands for one proof of the answer candidate. When one of these clauses is propositional, all query variables within this clause were bound and hence become bound in the entire formula.
2. Since a query variable can be bound to many constants, each representing a different query answer, the first-order lineage formula represents all these answers.

### 5.2 Probability Bounds for Lineage Formulas

In this section, we develop lower and upper bounds for the probability of any query answer that can be obtained from grounding a first-order lineage formula. We proceed by constructing two propositional lineage formulas $\phi_{low}$ and $\phi_{up}$ from a given first-order lineage formula $\Phi$. Later, the probabilities of $\phi_{low}$ and $\phi_{up}$ serve as lower and upper bounds on the probabilities of all query answers captured by $\Phi$. More formally, if $\phi_1, \ldots, \phi_n$ represent all query answers we would obtain by fully grounding $\Phi$, then it holds that:

$$\forall i \in \{1, \ldots, n\} : \; P(\phi_{low}) \leq P(\phi_i) \leq P(\phi_{up})$$

Building upon results of [28,49,55], we start by considering bounds for propositional formulas, from which we extend to the more general case of first-order lineage. Then, we show that these bounds converge monotonically to the probabilities $P(\phi_i)$ of each query answer $\phi_i$, as we continue to ground $\Phi$.

**Bounds for Propositional Lineage.** Following [49], we relate the probability of two propositional lineage formulas $\phi$ and $\psi$ via their sets of models $\mathcal{M}(\phi)$ and $\mathcal{M}(\psi)$ (see Equation (4)), i.e., the sets of possible worlds over which $\phi$ and $\psi$ evaluate to *true*.

**Proposition 7.** *For two propositional lineage formulas $\phi$ and $\psi$ it holds that:*

$$\mathcal{M}(\phi) \subseteq \mathcal{M}(\psi) \quad \Rightarrow \quad P(\phi) \leq P(\psi)$$

*Proof.*

$$\begin{aligned} P(\phi) &\overset{\text{Equation (5)}}{=} \textstyle\sum_{\mathcal{W} \in \mathcal{M}(\phi)} P(\mathcal{W}) \\ &\leq \textstyle\sum_{\mathcal{W} \in \mathcal{M}(\phi)} P(\mathcal{W}) + \sum_{\mathcal{W} \in \mathcal{M}(\psi) \setminus \mathcal{M}(\phi)} P(\mathcal{W}) \\ &\overset{\mathcal{M}(\phi) \subseteq \mathcal{M}(\psi)}{=} \textstyle\sum_{\mathcal{W} \in \mathcal{M}(\psi)} P(\mathcal{W}) \\ &\overset{\text{Equation (5)}}{=} P(\psi) \end{aligned}$$

Since we assume $\mathcal{M}(\phi) \subseteq \mathcal{M}(\psi)$, the possible worlds satisfying $\phi$ fulfill $\psi$ as well. However, there might be more worlds satisfying $\psi$ but not $\phi$. This might yield more terms over which the sum of Equation (5) ranges, and thus we obtain $P(\phi) \leq P(\psi)$.

*Example 34.* Consider the two propositional formulas $\phi \equiv I_1$ and $\psi \equiv I_1 \vee I_2$. From $\mathcal{M}(I_1) \subseteq \mathcal{M}(I_1 \vee I_2)$ it follows that $P(I_1) \leq P(I_1 \vee I_2)$, which we can easily verify by Equation (5).

To turn Proposition 7 into upper and lower bounds, we proceed by considering *conjunctive clauses* in the form of conjunctions of propositional literals. Then, following a result from [49], we obtain the following proposition.

**Proposition 8.** *Let $\phi$, $\psi$ be two propositional, conjunctive clauses. It holds that:*

$$\mathcal{M}(\phi) \subseteq \mathcal{M}(\psi) \quad \Leftrightarrow \quad \mathit{Tup}(\phi) \supseteq \mathit{Tup}(\psi)$$

The above statement expresses that adding literals to a conjunction $\phi$ removes satisfying worlds from $\mathcal{M}(\phi)$.

*Example 35.* For the two clauses $I_1 \wedge I_2$ and $I_1$ it holds that $\mathit{Tup}(I_1 \wedge I_2) \supseteq \mathit{Tup}(I_1)$ and thus Proposition 8 yields $\mathcal{M}(I_1 \wedge I_2) \subseteq \mathcal{M}(I_1)$.

We now establish a relationship between two formulas in Disjunctive Normal Form (DNF) (see Definition 12) via their conjunctive clauses as in [49,55]. Since any propositional formula can be transformed equivalently into DNF, this result is generally applicable.

**Lemma 3.** *For two propositional DNF formulas $\phi \equiv \phi_1 \vee \cdots \vee \phi_n$ and $\psi \equiv \psi_1 \vee \cdots \vee \psi_n$, it holds that:*

$$\forall \phi_i \exists \psi_j : \mathcal{M}(\phi_i) \subseteq \mathcal{M}(\psi_j) \Rightarrow \mathcal{M}(\phi) \subseteq \mathcal{M}(\psi)$$

If we can map all clauses $\phi_i$ of a formula $\phi$ to a clause $\psi_j$ of $\psi$ with more satisfying worlds, i.e., $\mathcal{M}(\phi_i) \subseteq \mathcal{M}(\psi_j)$, then $\psi$ has more satisfying worlds than $\phi$. This mapping of clauses is established via Proposition 8.

*Example 36.* For the propositional DNF formula $\phi \equiv (I_1 \wedge I_2) \vee (I_1 \wedge I_3) \vee I_4$, we can map each conjunctive clause in $\phi$ to a clause in $\psi \equiv I_1 \vee I_4$. Hence, $\psi$ has more models than $\phi$, i.e., $\mathcal{M}(\phi) \subseteq \mathcal{M}(\psi)$.

Thus, Lemma 3 enables us to compare the probabilities of propositional formulas in DNF based on their clause structure.

**Converting Formulas to DNF.** When transforming any propositional formula into DNF, we can first iteratively apply De Morgan's law [67] which pushes negations down in a formula:

$$\begin{aligned} \neg \textstyle\bigwedge_i \Phi_i &\equiv \textstyle\bigvee_i \neg \Phi_i \\ \neg \textstyle\bigvee_i \Phi_i &\equiv \textstyle\bigwedge_i \neg \Phi_i \end{aligned} \tag{17}$$

Thereafter, we apply the distributive law which allows the following observation.

**Observation 13** *If a tuple $I$ occurs exactly once in a propositional formula $\phi$, then all occurrences of $I$ in the DNF of $\phi$ have the same sign.*

The reason is that the sign of a tuple $I$ changes only when De Morgan's law is applied. However, when applying De Morgan's law, no tuples are duplicated. When utilizing the distributive law, tuples are duplicated but preserve their signs.

*Example 37.* Applying the distributive law to $(I_1 \vee I_2) \wedge \neg I_3$ yields $(I_1 \wedge \neg I_3) \vee (I_2 \wedge \neg I_3)$. Now, $I_3$ occurs twice, but its sign was preserved.

**Bounds for First-Order Lineage** For our following constructions on first-order formulas, we assume the first-order formulas to be given in PDNF. Next, given a first-order lineage formula $\Phi$, we construct two propositional formulas $\phi_{low}$ and $\phi_{up}$ whose probabilities then serve as lower and upper bound on $\Phi$, respectively.

**Definition 26.** *Let $\Phi$ be a first-order lineage formula.*

1. *We construct the propositional lineage formula $\phi_{up}$ by substituting every literal $R(\bar{X})$ in $\Phi$ with*
   - true *if $R(\bar{X})$ occurs positive in the PDNF of $\Phi$, or*
   - false *if $R(\bar{X})$ occurs negated in the PDNF of $\Phi$.*
2. *We construct the propositional lineage formula $\phi_{low}$ by substituting every literal $R(\bar{X})$ in $\Phi$ with*
   - false *if $R(\bar{X})$ occurs positive in the PDNF of $\Phi$, or*
   - true *if $R(\bar{X})$ occurs negated in the PDNF of $\Phi$.*

The idea of the above definition is as follows. If we replace a positive literal by *true*, we add models to the resulting formula. Hence, due to Proposition 7 the resulting formula can serve as an upper bound on the probability, which we show formally later. The remaining three cases are analogous. We note that $R$ can be intensional, extensional and even arithmetic.

*Example 38.* We consider Figure 4 and the first-order lineage formula:

$$\Phi := I_1 \wedge \exists X\, WonAward(X, BestPicture)$$

Then, the upper bound is given by $P(\phi_{up}) = P(I_1 \wedge true) = p(I_1) = 0.8$ and the lower bound is $P(\phi_{low}) = P(I_1 \wedge false) = P(false) = 0$. If we execute one SLD step (see Definition 25) on $\Phi$ we obtain $I_1 \wedge (I_9 \vee I_{10})$. Its probability is $P(I_1 \wedge (I_9 \vee I_{10}) = 0.8 \cdot (1 - (1 - 0.4) \cdot (1 - 0.9) = 0.752$ which is correctly captured by the upper and lower bound.

As a next step, we discuss the application of Definition 26 to general first-order lineage formulas which do not necessarily adhere any normal form.

**Proposition 9.** *By first exhaustively applying De Morgan's law of Equation* (17) *on a first-order lineage formula $\Phi$, we can apply Definition 26 to $\Phi$, even if $\Phi$ is not in PDNF. Hence, constructing $\phi_{up}$ and $\phi_{low}$ can be done in $O(|\Phi|)$.*

*Proof.* We can implement De Morgan by traversing the formula once, which thus is in $O(|\Phi|)$. Subsequently, we traverse the formula again and replace all first-order literals by *true* or *false* as devised in Definition 26. Observation 13 ensures the replacements to be unique for each literal.

**Convergence of Bounds.** Our last step is to show that, when constructing first-order lineage $\Phi$ (see Definition 25) for a fixed query answer $\phi$ resulting from $\Phi$, the probability bounds converge monotonically to the probability of the propositional lineage formula $P(\phi)$ with each SLD step.

**Theorem 2.** *Let $\Phi_1, \ldots, \Phi_n$ denote a series of first-order formulas obtained from iteratively grounding a conjunctive query via the form of SLD resolution provided in Definition 25 until we reach the propositional formula $\phi$. Then, rewriting each $\Phi_i$ to $\phi_{i,low}$ and $\phi_{i,up}$ according to Definition 26 creates a monotonic series of lower and upper bounds $P(\phi_{i,low})$, $P(\phi_{i,up})$ for the probability $P(\phi)$. That is:*

$$\begin{aligned} 0 \leq P(\phi_{1,low}) \leq \cdots \leq P(\phi_{n,low}) \leq P(\phi) \\ \leq P(\phi_{n,up}) \leq \cdots \leq P(\phi_{1,up}) \leq 1 \end{aligned}$$

*Proof.* The proof proceeds inductively over the structure of Definition 25, where we show that each SLD step preserves the bounds. We assume the observed literal $R(\bar{X})$ to occur positively in the PDNF of $\Phi_i$. The negated version is handled analogously.

1. We have an intensional literal $R(\bar{X})$ which was substituted in $\Phi_i$ by the disjunction of deduction rules' bodies, which we call $\Psi$ here, to yield $\Phi_{i+1}$. Because there are only literals and no tuple identifiers in $\Psi$, Definition 26 yields $\psi_{low} \equiv \mathit{false}$ and $\psi_{up} \equiv \mathit{true}$. Hence, the bounds of $\Phi_{i+1}$ are not altered, which reads as $P(\phi_{i+1,up}) = P(\phi_{i,up})$ and $P(\phi_{i+1,low}) = P(\phi_{i,low})$.
2. As in Definition 25, we separate the cases of different variables.
   (a) In this case we consider an extensional literal $R(\bar{X})$ where $\mathit{Var}(\bar{X})$ are query variables. Now, $\mathit{SLD}(R(\bar{X}), \Phi_i)$ delivers a set of formulas. Let $\Phi_{i+1}$ be an arbitrary formula in this set. We obtain $\Phi_{i+1}$ by replacing $R(\bar{X})$ in $\Phi_i$ by a tuple identifier $I$. Hence, in the DNF of $\phi_{i+1,up}$ we added $I$ to the clauses, whereas in the DNF of $\phi_{i,up}$ we replace $R(\bar{X})$ by *true*. Thus, Lemma 3 applies and we have $P(\phi_{i+1,up}) \leq P(\phi_{i,up})$. The reasoning for lower bounds is analogous.
   (b) Again, we have an extensional literal $R(\bar{X})$, but all variables $\mathit{Var}(\bar{X})$ are bound by an existential quantifier. As a result, each $\Phi_{i+1}$ in the set $\mathit{SLD}(R(\bar{X}, \Phi_i)$ is constructed from $\Phi_i$ by the first line of Equation (16). Now, the DNF of $\phi_{i,up}$ has clauses where $R(\bar{X})$ was substituted by *true*. Then, in $\phi_{i+1,up}$ each clause featuring a new tuple identifier $I$ can be mapped to one of these clauses in the DNF of $\phi_{i,up}$. Therefore, Lemma 3 gives us $P(\phi_{i+1,up}) \leq P(\phi_{i,up})$. Again, lower bounds are handled analogously.
   (c) If the variables $\bar{X}$ in the extensional literal $R(\bar{X})$ are universally quantified, then $R(\bar{X})$ in $\Phi_i$ is replaced by a conjunction (as given in the second line of Equation (16)) to yield $\Phi_{i+1}$. In the DNF of $\phi_{i,up}$, we employed *true* whenever $R(\bar{X})$ occurred. Conversely, in $\phi_{i+1,up}$ we replaced $R(\bar{X})$ by a conjunction of tuple identifiers. The resulting extended

clauses of $\phi_{i+1,up}$ can be mapped to a clause of $\phi_{i,up}$, so Lemma 3 applies: $P(\phi_{i+1,up}) \leq P(\phi_{i,up})$. The lower bounds are addressed analogously.

3. Here, a literal $R(\bar{X})$ was replaced in $\Phi_i$ by *false* to yield $\Phi_{i+1}$. Hence, for the lower bounds constructed according to Definition 26, we have $P(\phi_{i,low}) = P(\phi_{i+1,low})$. For the upper bounds Lemma 3 delivers $P(\phi_{i+1,up}) \leq P(\phi_{i,up})$, since the PDNF of $\Phi_{i+1}$ has fewer clauses as the PDNF of $\Phi_i$.
4. In the last case, $R(\bar{X})$ is arithmetic and $\bar{X}$ consists of constants only. Now, if $R(\bar{X})$ evaluates to *true*, we have $\phi_{i,up} = \phi_{i+1,up}$ and hence also $P(\phi_{i,up}) = P(\phi_{i+1,up})$. For the lower bound, the DNF of $\phi_{i+1,low}$ can have more clauses than the DNF of $\phi_{i,low}$ and so Lemma 3 comes to our rescue again: $P(\phi_{i,low}) \leq P(\phi_{i+1,low})$. Conversely, if $R(\bar{X})$ evaluates to *false*, the reasoning for the upper and lower bounds is inverted.

The resulting lower and upper bounds for all answer candidates can be plugged into any top-$k$ algorithm (see [36] for an extensive overview) that—in our case—will then iteratively refine these lower and upper bounds via SLD resolution until a termination condition is reached. The seminal line of threshold algorithms proposed by Fagin, Lotem and Naor [26], for example, iteratively maintains two disjoint sets of top-$k$ answers and remaining answer candidates, coined *top-k candidates*, respectively, and it terminates when:

$$\min\{P(\phi_{i,low}) \mid \phi_i \in \textit{top-k}\} \geq \max\{P(\phi_{i,up}) \mid \phi_i \in \textit{candidates}\}$$

## 6 Learning Tuple Probabilities

Most works in the context of PDBs assume the database tuples along with their probabilities to be given as input. Also the preceding sections of this chapter followed this route. Nevertheless, when creating, updating or cleaning a PDB, the tuple probabilities have to be altered or even be newly created—in other words: they have to be *learned*. Learning the probability values of databases tuples from labeled lineage formulas thus is the subject of the present section and is also discussed in more detail in [24].

*Example 39.* Our running example resembles the information-extraction setting of Section 1.1, in which we employ a set of textual patterns to extract facts from various Web domains. However, instead of knowing all probabilities of all tuples the respective values in the *UsingPattern* and *FromDomain* relations are missing as indicated by the question marks in Figure 5. We thus are unsure about the reliability—or "trustworthiness"—of the textual patterns and the Web domains that led to the extraction of our remaining facts, respectively. Grounding the deduction rules of Equation (1) and Equation (2) against the database tuples of Figure 5 yields the new tuples *BornIn(Spielberg, Cinncinati)*, *BornIn(Spielberg, LosAngeles)*, and *WonPrize(Spielberg,AcademyAward)*. Figure 6 shows these new tuples along with their propositional lineage formulas.

WonPrizeExtraction

| | Subject | Object | Pid | Did | $p$ |
|---|---|---|---|---|---|
| $I_1$ | *Spielberg* | *AcademyAward* | 1 | 1 | 0.6 |
| $I_2$ | *Spielberg* | *AcademyAward* | 2 | 1 | 0.3 |

BornInExtraction

| | Subject | Object | Pid | Did | $p$ |
|---|---|---|---|---|---|
| $I_3$ | *Spielberg* | *Cinncinati* | 3 | 1 | 0.7 |
| $I_4$ | *Spielberg* | *LosAngeles* | 3 | 2 | 0.4 |

UsingPattern

| | Pid | Pattern | $p$ |
|---|---|---|---|
| $I_5$ | 1 | *Received* | ? |
| $I_6$ | 2 | *Won* | ? |
| $I_7$ | 3 | *Born* | ? |

FromDomain

| | Did | Domain | $p$ |
|---|---|---|---|
| $I_8$ | 1 | *Wikipedia.org* | ? |
| $I_9$ | 2 | *Imdb.com* | ? |

**Fig. 5.** Example Probabilistic Database with Missing Probability Values

A closer look at the new tuples reveals, however, that not all of them are correct. For instance, *BornIn(Spielberg,LosAngeles)* is wrong, so we might rather label it with the probability of 0.0. Moreover, *WonPrize(Spielberg,AcademyAward)* is likely correct, but we are unsure, hence we label it with the probability of 0.7, as shown on top of Figure 6. Given the probability labels of the query answers, the goal of the learning procedure is to learn the database tuples' unknown probability values for *UsingPattern* and *FromDomain*, such that the lineage formulas again produce the given probability labels. The probabilities of the tuples of *WonPrizeExtraction* and *BornInExtraction*, on the other hand, should remain unchanged.

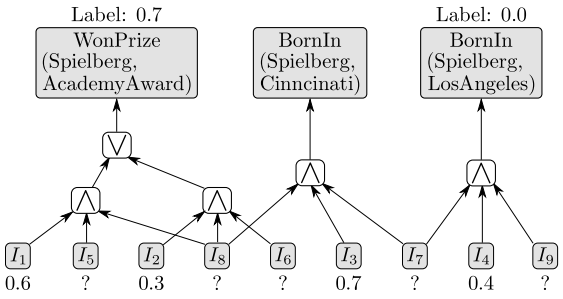



**Fig. 6.** Partially Labeled Lineage Formulas

### 6.1 Encoding Propositional Lineage into Polynomials

For the theoretical analysis of the learning problem presented in this section, we devise an alternative way of computing probabilities of lineage formulas via polynomial expressions. First, we reduce the number of terms in the sum of Equation (5) by considering just tuples $\mathit{Tup}(\phi)$ that occur in the propositional lineage formula $\phi$.

**Proposition 10.** *We can compute $P(\phi)$ relying on tuples in $\mathit{Tup}(\phi)$, only, by writing:*

$$P(\phi) = \sum_{\mathcal{V} \in \mathcal{M}(\phi,\, \mathit{Tup}(\phi))} \underbrace{P(\mathcal{V},\, \mathit{Tup}(\phi))}_{\textit{Definition 10}} \tag{18}$$

*Proof.*

$$\begin{aligned} P(\phi) &= \textstyle\sum_{\mathcal{W} \in \mathcal{M}(\phi, \mathcal{T})} P(\mathcal{W}, \mathcal{T}) \\ &= \Big(\textstyle\sum_{\mathcal{V} \in \mathcal{M}(\phi,\, \mathit{Tup}(\phi))} P(\mathcal{V},\, \mathit{Tup}(\phi))\Big) \cdot \underbrace{\Big(\textstyle\sum_{\mathcal{V} \subseteq (\mathcal{T} \setminus \mathit{Tup}(\phi))} P(\mathcal{V}, \mathcal{T} \setminus \mathit{Tup}(\phi))\Big)}_{=1 \text{ by Proposition 1}} \end{aligned}$$

Thus, Equation (18) expresses $P(\phi)$ as a polynomial. Its terms are defined as in the third item of Definition 10, and the variables are $p(I)$ for $I \in \mathit{Tup}(\phi)$. The degree of the polynomial is limited as follows.

**Corollary 1.** *The probability $P(\phi)$ of a propositional lineage formula $\phi$ can be expressed by a multi-linear polynomial over variables $p(I)$, for $I \in \mathit{Tup}(\phi)$, with a degree of at most $|\mathit{Tup}(\phi)|$.*

*Proof.* By inspecting Proposition 10, we note that the sum ranges over subsets of $\mathit{Tup}(\phi)$ only, hence each term has a degree of at most $|\mathit{Tup}(\phi)|$.

*Example 40.* Considering the propositional lineage formula $\phi \equiv I_1 \vee I_2$, the occurring tuples are $\mathit{Tup}(\phi) = \{I_1, I_2\}$. Then, it holds that $\{I_1, I_2\} \models \phi$, $\{I_1\} \models \phi$, and $\{I_2\} \models \phi$. Hence, we can write $P(\phi) = p(I_1) \cdot p(I_2) + p(I_1) \cdot (1 - p(I_2)) + (1 - p(I_1)) \cdot p(I_2)$. Thus, $P(\phi)$ is a polynomial over the variables $p(I_1)$, $p(I_2)$ and has degree $2 = |\mathit{Tup}(\phi)| = |\{I_1, I_2\}|$.

### 6.2 Learning Problem

We now move away from the case where the probability values of all database tuples are known, which was a basic assumption we made for the previous sections. Instead, we intend to learn the unknown probability values of (some of) these tuples (e.g., of $I_5$–$I_9$ in Example 39). More formally, for a tuple-independent PDB $(\mathcal{T}, p)$, we consider $\mathcal{T}_l \subseteq \mathcal{T}$ to be the set of base tuples for which we learn their probability values. That is, initially $p(I)$ is unknown for all $I \in \mathcal{T}_l$. Conversely, $p(I)$ is known and fixed for all $I \in \mathcal{T} \setminus \mathcal{T}_l$. To be able to complete $p(I)$, we are given labels in the form of pairs $(\phi_i, l_i)$, each containing a propositional lineage formula $\phi_i$ (i.e., a query answer) and its desired probability $l_i$. We formally define the resulting learning problem as follows.

**Definition 27.** *We are given a probabilistic database $(\mathcal{T}, p)$, a set of tuples $\mathcal{T}_l \subseteq \mathcal{T}$ with unknown probability values $p(I_l)$ and a multi-set of given labels $\mathcal{L} = \langle(\phi_1, l_1), \ldots, (\phi_n, l_n)\rangle$, where each $\phi_i$ is a propositional lineage formula over $\mathcal{T}$ and each $l_i \in [0, 1] \subset \mathbb{R}$ is a probability for $\phi_i$. Then, the* learning problem *is defined as follows:*

$$\begin{array}{ll} \textit{Determine:} & p(I_l) \in [0,1] \subset \mathbb{R} \textit{ for all } I_l \in \mathcal{T}_l \\ \textit{such that:} & P(\phi_i) = l_i \textit{ for all } (\phi_i, l_i) \in \mathcal{L} \end{array}$$

Intuitively, we aim to set the probability values of the base tuples $I_l \in \mathcal{T}_l$ such that the labeled lineage formulas $\phi_i$ again yield the probability $l_i$. We want to remark that all probability values of tuples in $\mathcal{T} \backslash \mathcal{T}_l$ remain unaltered. Also, we note that the Boolean labels *true* and *false* can be represented as $l_i = 0.0$ and $l_i = 1.0$, respectively. Hence, Boolean labels resolve to a special case of the labels of Definition 27.

*Example 41.* Formalizing the problem setting of Example 39, we obtain $\mathcal{T} := \{I_1, \ldots, I_9\}$, $\mathcal{T}_l := \{I_5, \ldots, I_9\}$ with labels $((I_1 \wedge I_5 \wedge I_8) \vee (I_2 \wedge I_6 \wedge I_8), 0.7)$, and $((I_3 \wedge I_7 \wedge I_9), 0.0)$.

### 6.3 Properties of the Learning Problem

We next discuss the complexity of solving the learning problem. Unfortunately, it exhibits hard instances. First, computing $P(\phi_i)$ may be $\#\mathcal{P}$-hard (see Lemma 2), which would require many Shannon expansions to compute an exact probability $P(\phi_i)$. But even for cases when all $P(\phi_i)$ can be computed in polynomial time (i.e., when Equation (6) is applicable), there are combinatorially hard cases of the above learning problem.

**Lemma 4.** *For a given instance of the learning problem of Definition 27, where all $P(\phi_i)$ with $(\phi_i, l_i) \in \mathcal{L}$ can be computed in polynomial time, deciding whether there exists a solution to the learning problem is $\mathcal{NP}$-hard.*

*Proof.* We encode the 3-Satisfiability Problem (3SAT) [31] for a Boolean formula $\psi \equiv \psi_1 \wedge \cdots \wedge \psi_n$ in Conjunctive Normal Form (CNF) into the learning problem of Definition 27. For each variable $X_i \in \mathit{Var}(\psi)$, we create two tuples $I_i$, $I_i'$ whose probability values will be learned. Hence, $2 \cdot |\mathit{Var}(\psi)| = |\mathcal{T}_l| = |\mathcal{T}|$. Then, for each $X_i$, we add the label $((I_i \wedge I_i') \vee (\neg I_i \wedge \neg I_i'), 1.0)$. The corresponding polynomial equation $p(I_i) \cdot p(I_i') + (1 - p(I_i)) \cdot (1 - p(I_i')) = 1.0$ has exactly two possible solutions for $p(I_i), p(I_i') \in [0, 1]$, namely $p(I_i) = p(I_i') = 1.0$ and $p(I_i) = p(I_i') = 0.0$. Next, we replace all variables $X_i$ in $\psi$ by their tuple $I_i$. Now, for each clause $\psi_i$ of $\psi$, we introduce one label $(\psi_i, 1.0)$. Altogether, we have $|\mathcal{L}| = |\mathit{Var}(\psi)| + n$ labels for the problem of Definition 27. Each labeled lineage formula $\phi$ has at most three variables, hence $P(\phi)$ takes at most 8 steps. Still, Definition 27 solves 3SAT, where the learned values of each pair of $p(I_i)$, $p(I_i')$ (either 0.0 or 1.0) correspond to a truth value of all $X_i$ for a satisfying assignment of $\psi$. From this, it follows that the decision problem formulated in Lemma 4 is $\mathcal{NP}$-hard.

After discussing the complexity of the learning problem, we characterize its solutions. First, there might also be *inconsistent* instances of the learning problem. That is, it may be impossible to define $p : \mathcal{T}_l \to [0, 1]$ such that all labels are satisfied.

*Example 42.* If we consider $\mathcal{T}_l := \{I_1, I_2\}$ with the labels $\mathcal{L} := \langle (I_1, 0.2), (I_2, 0.3), (I_1 \wedge I_2, 0.9) \rangle$, then it is impossible to fulfill all three labels at the same time.

From a practical point of view, there remain a number of questions regarding Definition 27. First, how many labels do we need in comparison to the number of tuples for which we are learning the probability values (i.e., $|\mathcal{L}|$ vs. $|\mathcal{T}_l|$)? And second, is there a difference in labeling lineage formulas that involve many tuples or very few tuples (i.e., $|\mathit{Tup}(\phi_i)|$)? These questions are addressed by the following theorem. It is based on the computation of probabilities of lineage formulas via their polynomial representation as in Corollary 1. We write the conditions of the learning problem $P(\phi_i) = l_i$ as polynomials over variables $p(I_l)$ of the form $P(\phi_i) - l_i$, where $I_l \in \mathcal{T}_l$ and the probability values $p(I)$ for all $I \in \mathcal{T} \backslash \mathcal{T}_l$ are fixed and hence represent constants.

**Theorem 3.** *If the labeling is consistent, the problem instances of Definition 27 can be classified as follows:*

1. *If $|\mathcal{L}| < |\mathcal{T}_l|$, the problem has infinitely many solutions.*
2. *If $|\mathcal{L}| = |\mathcal{T}_l|$ and the polynomials $P(\phi_i) - l_i$ have common zeros, then the problem has infinitely many solutions.*
3. *If $|\mathcal{L}| = |\mathcal{T}_l|$ and the polynomials $P(\phi_i) - l_i$ have no common zeros, then the problem has at most $\prod_i |\mathit{Tup}(\phi_i) \cap \mathcal{T}_l|$ solutions.*
4. *If $|\mathcal{L}| > |\mathcal{T}_l|$, then the polynomials $P(\phi_i) - l_i$ have common zeros, thus reducing this to one of the previous cases.*

*Proof.* The first case is a classical under-determined system of equations. In the second case, without loss of generality, there are two polynomials $P(\phi_i) - l_i$ and $P(\phi_j) - l_j$ with a common zero, say $p(I_k) = c_k$. Setting $p(I_k) = c_k$ satisfies both $P(\phi_i) - l_i = 0$ and $P(\phi_j) - l_j = 0$, hence we have $\mathcal{L}' := \mathcal{L} \backslash \langle (\phi_i, l_i), (\phi_j, l_j) \rangle$ and $\mathcal{T}_l' := \mathcal{T}_l \backslash \{I_k\}$ which yields the first case of the theorem again ($|\mathcal{L}'| < |\mathcal{T}_l'|$). Regarding the third case, Bezout's theorem [20], a central result from algebraic geometry, is applicable: for a system of polynomial equations, the number of solutions (including their multiplicities) over variables in $\mathbb{C}$ is equal to the product of the degrees of the polynomials. In our case, the polynomials are $P(\phi_i) - l_i$ with variables $p(I_l)$ where $I_l \in \mathcal{T}_l$. So, according to Corollary 1 their degree is at most $|\mathit{Tup}(\phi_i) \cap \mathcal{T}_l|$. Since our variables $p(I_l)$ range only over $[0, 1] \subset \mathbb{R}$, and Corollary 1 is an upper bound only, $\prod_i |\mathit{Tup}(\phi_i) \cap \mathcal{T}_l|$ is an upper bound on the number of solutions. In the fourth case, the system of equations is over-determined, such that redundancies like common zeros reduces the problem to one of the previous cases.

*Example 43.* We illustrate the theorem by providing examples for each of the four cases.

1. In Example 41's formalization of Example 39, we have $|\mathcal{T}_l| = 5$ and $|\mathcal{L}| = 2$. So, the problem is under-specified and has infinitely many solutions, since assigning $p(I_7) = 0.0$ enables $p(I_9)$ to take any value in $[0, 1] \subset \mathbb{R}$.
2. We assume $\mathcal{T}_l = \{I_5, I_6, I_7\}$, and $\mathcal{L} = \langle (I_5 \wedge \neg I_6, 0.0), (I_5 \wedge \neg I_6 \wedge I_7, 0.0), (I_5 \wedge I_7, 0.0) \rangle$. This results in the equations $p(I_5) \cdot (1 - p(I_6)) = 0.0$, $p(I_5) \cdot (1 - p(I_6)) \cdot p(I_7) = 0.0$, and $p(I_5) \cdot p(I_7) = 0.0$, where $p(I_5)$ is a common zero to all three polynomials. Hence, setting $p(I_5) = 0.0$ allows $p(I_6)$ and $p(I_7)$ to take any value in $[0, 1] \subset \mathbb{R}$.
3. Let us consider $\mathcal{T}_l = \{I_7, I_8\}$.
   (a) If $\mathcal{L} = \langle (I_7, 0.4), (I_8, 0.7) \rangle$, then there is exactly one solution as predicted by $|\mathit{Tup}(I_7)| \cdot |\mathit{Tup}(I_8)| = 1$.
   (b) If $\mathcal{L} = \langle (I_7 \wedge I_8, 0.1), (I_7 \vee I_8, 0.6) \rangle$, then there are two solutions, namely $p(I_7) = 0.2$, $p(I_8) = 0.5$ and $p(I_7) = 0.5$, $p(I_8) = 0.2$. Here, $\prod_i |\mathit{Tup}(\phi_i) \cap \mathcal{T}_l| = |\mathit{Tup}(I_7 \wedge I_8)| \cdot |\mathit{Tup}(I_7 \vee I_8)| = 4$ is an upper bound.
4. We extend the second case of this example by the label $(I_5, 0.0)$, thus yielding the same solutions but having $|\mathcal{L}| > |\mathcal{T}_l|$.

In general, a learning problem instance has many solutions, where Definition 27 does not specify a precedence, but all of them are equivalent. The number of solutions shrinks by adding labels to $\mathcal{L}$, or by labeling lineage formulas $\phi_i$ that involve fewer tuples in $\mathcal{T}_l$ (thus resulting in a smaller intersection $|\mathit{Tup}(\phi_i) \cap \mathcal{T}_l|$). Hence, to achieve more uniquely specified probabilities for all tuples $I_l \in \mathcal{T}_l$, in practice we should obtain the same number of labels as the number of tuples for which we learn their probability values, i.e., $|\mathcal{L}| = |\mathcal{T}_l|$, and label those lineage formulas with fewer tuples in $\mathcal{T}_l$.

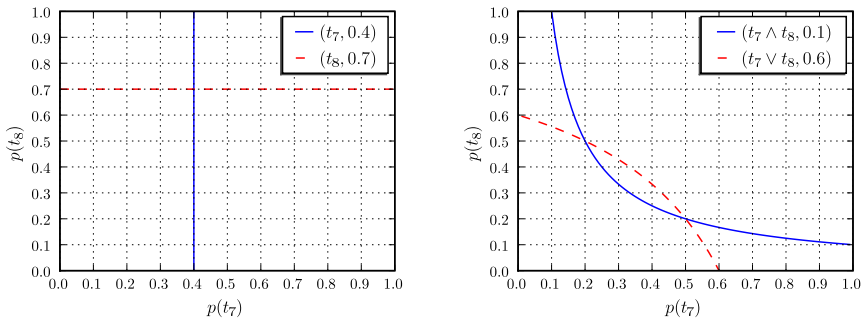
Now that we characterized the number of solutions, we furthermore provide an insight on their nature. We give conditions on learning problems which imply the existence of an integer solution, i.e., that assigns only 0 or 1 as tuple probabilities. Hence, the resulting tuples are either non-existent or deterministic as in conventional databases.

**Proposition 11.** *For a learning problem, where*

1. $\forall I \in \mathcal{T} \backslash \mathcal{T}_l : p(I) \in \{0, 1\}$
2. $(\phi_i, l_i) \in \mathcal{L} : l_i \in \{0, 1\}$
3. $\bigwedge_{(\phi_i,1)\in\mathcal{L}} \phi_i \ \wedge \bigwedge_{(\phi_i,0)\in\mathcal{L}} \neg\phi_i$ *is satisfiable,*

*there exists an* integer solution $p'$*, that is for all* $I_l \in \mathcal{T}_l : p'(I_l) \in \{0, 1\}$.

*Proof.* Due to the first requirement we can remove all tuples in $\mathcal{T} \backslash \mathcal{T}_l$ from the labels' formulas $\phi$, since these tuples correspond to either *true* or *false*. Likewise, the second condition allows the construction of the formula $\bigwedge_{(\phi_i,1)\in\mathcal{L}} \phi_i \ \wedge \bigwedge_{(\phi_i,0)\in\mathcal{L}} \neg\phi_i$. As we require the existence of a satisfying assignment for this formula, precisely this assignment is the integer solution.




(a) Labels of case 3(a) of Example 43 (b) Labels of case 3(b) Example 43

**Fig. 7.** Visualization of the Learning Problem

**Visual Interpretation.** Based on algebraic geometry, the learning problem allows for a visual interpretation. All possible definitions of probability values for tuples in $\mathcal{T}_l$, that is, $p : \mathcal{T}_l \to [0, 1]$, span the hypercube $[0, 1]^{|\mathcal{T}_l|}$. In Example 43, cases 3(a) and 3(b), the hypercube has two dimensions, namely $p(I_7)$ and $p(I_8)$, as depicted in Figures 7(a) and 7(b). Hence, one definition of $p$ specifies exactly one point in the hypercube. Moreover, all definitions of $p$ that satisfy a given label define a curve (or plane) through the hypercube (e.g., the two labels in Figure 7(a) define two straight lines). Also, the points, in which all labels' curves intersect, represent solutions to the learning problem (e.g., the solutions of Example 43, case 3(b), are the intersections in Figure 7(b)). If the learning problem is inconsistent, there is no point in which all labels' curves intersect. Furthermore, if the learning problem has infinitely many solutions, the labels' curves intersect in curves or planes, rather than points.

### 6.4 Gradient Based Solutions

We formally characterized the learning problem and devised the basic properties of its solutions. From a visual perspective, Definition 27 established curves and planes whose intersections represent the solutions (see, e.g., Figure 7(b)). We now introduce different objective functions that describe surfaces whose optima correspond to these solutions. For instance, the problem of Figure 7(b) has the surface of Figure 8(a) if we the employ mean squared error (MSE) as the objective, which will be defined in this section. Calculating a gradient on such a surface thus allows the application of an optimization method to solve the learning problem.

**Desired Properties.** Before we define objective functions for solving the learning problem, we establish a list of desired properties of these (which we do not

claim to be complete). Later, we judge different objectives based on these properties.

**Definition 28.** *An objective function to the learning problem should satisfy the following three* desired properties*:*

1. *All instances of the learning problem of Definition 27 can be expressed, including inconsistent ones.*
2. *If all $P(\phi_i)$ are computable in polynomial time, then also the objective is computable in polynomial time.*
3. *The objective is stable, that is $\mathcal{L} := \langle(\phi_1, l_1), \ldots, (\phi_n, l_n)\rangle$ and $\mathcal{L} \cup \langle(\phi_i', l_i)\rangle$ with $\phi_i' \equiv \phi_i$, $(\phi_i, l_i) \in \mathcal{L}$ define the same surface.*

Here, the first case ensures that the objective can be applied to all instances of the learning problem. We insist on including inconsistent instances, because they occur often in practice. The second property restricts a blow-up in computation, which yields the following useful characteristic: if we can compute $P(\phi)$ for all labels, e.g., for labeled query answers, then we can also compute the objective function. Finally, the last of the desiderata reflects an objective function's ability to detect dependencies between labels. Since $\phi_i \equiv \phi_i'$ both $\mathcal{L}$ and $\mathcal{L} \cup \langle(\phi_i', l_i)\rangle$ allow exactly the same solutions, the surface should be the same. Unfortunately, including convexity of an objective as an additional desired property is not possible. For example Figure 7(b) has two disconnected solutions, which induce at least two optima, thus prohibiting convexity. In the following, we establish two objective functions, which behave very differently with respect to the desired properties.

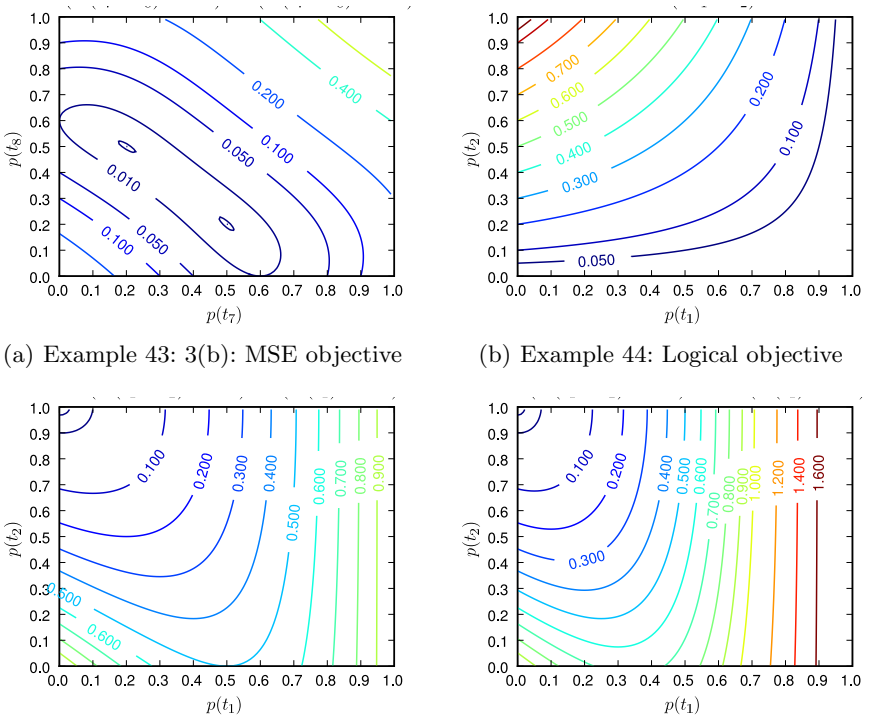
**Logical Objective.** If we restrict the probability labels of the learning problem to $l_i \in \{0.0, 1.0\}$, we can define an objective function based on computing probabilities of lineage formulas as follows.

**Definition 29.** *Let an instance of the learning problem of Definition 27 be given by a probabilistic database $(\mathcal{T}, p)$, tuples with unknown probability values $\mathcal{T}_l \subseteq \mathcal{T}$, and labels $\mathcal{L} = \langle(\phi_1, l_1), \ldots, (\phi_n, l_n)\rangle$ such that all $l_i \in \{0.0, 1.0\}$. Then, the* logical objective *is formulated as follows:*

$$\mathit{Logical}(\mathcal{L}, p) := P\left(\bigwedge_{(\phi_i, l_i) \in \mathcal{L}, l_i = 1.0} \phi_i \;\wedge \bigwedge_{(\phi_i, l_i) \in \mathcal{L}, l_i = 0.0} \neg\phi_i\right) \tag{19}$$

The above definition is a maximization problem, and its global optima are identified by $\mathit{Logical}(\mathcal{L}, p) = 1.0$. Moreover, from Definition 13, we may obtain its derivative.

*Example 44.* Let $\mathcal{T} = \mathcal{T}_l := \{I_1, I_2\}$ and $\mathcal{L} := \langle(I_1 \vee I_2, 1.0), (I_1, 0.0)\rangle$ be given. Then, $\mathit{Logical}(\mathcal{L}, p)$ is instantiated as $P((I_1 \vee I_2) \wedge \neg I_1) = P(\neg I_1 \wedge I_2)$. Visually, this defines a surface whose optimum lies in $p(I_1) = 0.0$ and $p(I_2) = 1.0$, as shown in Figure 8(b).




(a) Example 43: 3(b): MSE objective

(b) Example 44: Logical objective

(c) Example 46: MSE objective

(d) Example 46: MSE objective, unstable

**Fig. 8.** Visualization of the MSE and Logical Objective Functions

With respect to Definition 28, the third desired property is fulfilled, as $P(\phi_i' \wedge \phi_i) = P(\phi_i)$. Hence, the surface of the logical objective, shown for instance in Figure 8(b), is never altered by adding equivalent labels. Still, the first property is not given, since the probability labels are restricted to $l_i \in \{0.0, 1.0\}$ and inconsistent problem instances collapse Equation (19) to $P(\mathit{false})$, thus rendering the objective non-applicable. Also, the second property is violated, because in the spirit of the proof of Lemma 4, we can construct an instance where for each label $P(\phi_i)$ on its own is computable in polynomial time, whereas the computation of the probability for Equation (19) is again $\#\mathcal{P}$-hard.

**Mean Squared Error Objective.** Another approach, which is also common in machine learning, lies in using the mean squared error (MSE) to define the objective function.

**Definition 30.** *Let an instance of the learning problem of Definition 27 be given by a probabilistic database $(\mathcal{T}, p)$, tuples with unknown probability values $\mathcal{T}_l \subseteq \mathcal{T}$, and labels $\mathcal{L} = \langle(\phi_1, l_1), \ldots, (\phi_n, l_n)\rangle$. Then, the* mean squared error objective *is*

*formulated as:*

$$MSE(\mathcal{L}, p) := \frac{1}{|\mathcal{L}|} \sum_{(\phi_i, l_i) \in \mathcal{L}} (P(\phi_i) - l_i)^2$$

*Moreover, its partial derivative with respect to the probability value $p(I)$ of the tuple is:*

$$\frac{\partial MSE(\mathcal{L}, p)}{\partial p(I)} := \frac{1}{|\mathcal{L}|} \sum_{(\phi_i, l_i) \in \mathcal{L}, I \in Tup(\phi_i)} 2 \cdot (P(\phi_i) - l_i) \cdot \underbrace{\frac{\partial P(\phi_i)}{\partial p(I)}}_{Definition\ 13}$$

The above formulation is a minimization problem whose solutions have 0.0 as the target value of the objective function.

*Example 45.* Example 43, case 3(b), is visualized in Figure 7(b). The corresponding surface induced by the MSE objective is depicted in Figure 8(a) and has its minima at the solutions of the learning problem.

Judging the above objective by means of Definition 28, we realize that the first property is met, as there are no restrictions on the learning problem, and inconsistent instances can be tackled (but deliver objective values larger than zero). Furthermore, since the $P(\phi_i)$'s occur in separate terms of the sum of the objective, the second desired property is maintained. However, the third desired property is violated, as illustrated by the following example.

*Example 46.* In accordance to Example 44 and Figure 8(b), we set $\mathcal{T} = \mathcal{T}_l := \{I_1, I_2\}$ and $\mathcal{L} := \langle (I_1 \vee I_2, 1.0), (I_1, 0.0) \rangle$. Then, the MSE objective defines the surface in Figure 8(c). However, if we replicate the label $(I_1, 0.0)$, thus resulting in Figure 8(d) (note the "times two" in the objective), its surface becomes steeper along the $p(I_1)$-axis, but has the same minimum. Thus, MSE's surface is not stable. Instead, it becomes more ill-conditioned *[45]*.

**Discussion.** Both the logical objective and the MSE objective have optima exactly at the solutions of the learning problem of Definition 27. With respect to the desired properties of Definition 28, we summarize the behavior of both objectives in the following table:

| | Properties | | |
|---|---|---|---|
| Objective | 1. | 2. | 3. |
| Logical | × | × | ✓ |
| MSE | ✓ | ✓ | × |

The two objectives satisfy opposing desired properties, and it is certainly possible to define other objectives behaving similarly to one of them. Unfortunately, there is little hope for an objective that will be adhering to all three properties. The second property inhibits computational hardness. However, Lemma 4 and the third property's logical tautology checking (i.e., $\models \phi_i \leftrightarrow \phi_i'$, which is *co-*$\mathcal{NP}$-complete) require this. In this regard the logical objective addresses both computationally hard problems by computing probabilities, whereas the MSE objective avoids the latter form of tautology checking.

## 7 Conclusions

In recent years, the need to efficiently manage large amounts of uncertain data has become evident as more and more data arise from various applications such as information extraction, sensor networks, and scientific data management. As a result, PDBs have evolved as an establish field of research in recent years [65]. In this chapter, we provide an overview of the key concepts of PDBs and the main challenges than need to be addressed. Specifically, we begin by describing the main characteristics of probabilistic databases assuming tuple independence, and we present the respective data model and query evaluation strategies. Apart from being uncertain, data can be annotated by other dimensions such as time and location. In this regard, we describe a closed and complete temporal-probabilistic database model [23], coined TPDB, which allows us to cope with data that is variable over time as well as uncertain. Complementary to the basics, we review state-of-the-art methods in this field and also describe some of our own recent research results including a top-$k$ style evaluation strategy [25]. The latter attempts to tackle the increased complexity of the probability computation step involved in query evaluation in PDBs. This is achieved by pruning answer candidates without fully grounding the lineage formula and hence saving also on the data computation step. Last, although most works assume the probabilities are provided as input along with the data, this assumption often does not hold and a learning approach is required. As we consider learning to be a key building block for future probabilistic database engines, we conclude this chapter by discussing such a learning approach [24] for creating, updating and cleaning of PDBs.

**Acknowledgements.** This article is based on the doctoral dissertation by Maximilian Dylla: "Efficient Querying and Learning in Probabilistic and Temporal Databases", Saarland University, February 2014 [22].

## References

1. S. Abiteboul, L. Herr, and J. V. den Bussche. Temporal Connectives versus Explicit Timestamps in Temporal Query Languages. In J. Clifford and A. Tuzhilin, editors, *Recent Advances in Temporal Databases, Proceedings of the International Workshop on Temporal Databases, Zürich, Switzerland, 17-18 September 1995*, Workshops in Computing, pages 43–57, Berlin, Heidelberg, New York, 1995. Springer.
2. S. Abiteboul, R. Hull, and V. Vianu, editors. *Foundations of Databases*. Addison-Wesley, Boston, MA, USA, 1st edition, 1995.
3. S. Abiteboul, P. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. *SIGMOD Record*, 16(3):34–48, Dec. 1987.
4. J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, Nov. 1983.
5. L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and Simple Relational Processing of Uncertain Data. In *Proceedings of the 24th International Conference on Data Engineering*, ICDE, pages 983–992, Washington, DC, USA, 2008. IEEE Computer Society.

6. S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives. DBpedia: A Nucleus for a Web of Open Data. In *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007*, volume 4825 of *Lecture Notes in Computer Science*, pages 722–735. Springer, 2007.
7. O. Benjelloun, A. Das Sarma, A. Halevy, M. Theobald, and J. Widom. Databases with uncertainty and lineage. *VLDB Journal*, 17(2):243–264, Mar. 2008.
8. J. Boulos, N. Dalvi, B. Mandhani, S. Mathur, C. Re, and D. Suciu. MYSTIQ: a system for finding more answers by using probabilities. In *Proceedings of the International Conference on Management of Data*, SIGMOD, pages 891–893, New York, NY, USA, 2005. ACM.
9. S. Brin. Extracting Patterns and Relations from the World Wide Web. In *Selected Papers from the International Workshop on The World Wide Web and Databases*, WebDB, pages 172–183, Berlin, Heidelberg, New York, 1999. Springer.
10. S. Ceri, G. Gottlob, and L. Tanca. What You Always Wanted to Know About Datalog (And Never Dared to Ask). *IEEE Transactions on Knowledge and Data Engineering*, 1(1):146–166, Mar. 1989.
11. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, USA, 3rd edition, 2009.
12. Y. Cui, J. Widom, and J. L. Wiener. Tracing the Lineage of View Data in a Warehousing Environment. *ACM Transactions on Database Systems*, 25(2):179–227, June 2000.
13. N. Dalvi, C. Ré, and D. Suciu. Probabilistic Databases: Diamonds in the Dirt. *Communications of the ACM*, 52(7):86–94, July 2009.
14. N. Dalvi, K. Schnaitter, and D. Suciu. Computing Query Probability with Incidence Algebras. In *Proceedings of the Twenty-ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS, pages 203–214, New York, NY, USA, 2010. ACM.
15. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDB Journal*, 16(4):523–544, Oct. 2007.
16. N. Dalvi and D. Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS, pages 293–302, New York, NY, USA, 2007. ACM.
17. N. Dalvi and D. Suciu. The Dichotomy of Probabilistic Inference for Unions of Conjunctive Queries. *Journal of the ACM*, 59(6):30:1–30:87, Jan. 2013.
18. N. N. Dalvi, K. Schnaitter, and D. Suciu. Computing query probability with incidence algebras. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2010, June 6-11, 2010, Indianapolis, Indiana, USA*, pages 203–214, New York, NY, USA, 2010. ACM.
19. N. N. Dalvi and D. Suciu. The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM*, 59(6):30, 2012.
20. A. Dickenstein and I. Z. Emiris. *Solving Polynomial Equations: Foundations, Algorithms, and Applications*. Springer, Berlin, Heidelberg, New York, 1st edition, 2010.
21. A. Dignös, M. H. Böhlen, and J. Gamper. Temporal alignment. In *Proceedings of the International Conference on Management of Data*, SIGMOD, pages 433–444, New York, NY, USA, 2012. ACM.
22. M. Dylla. *Efficient Querying and Learning in Probabilistic and Temporal Databases*. PhD thesis, Saarland University, 2014.

23. M. Dylla, I. Miliaraki, and M. Theobald. A Temporal-Probabilistic Database Model for Information Extraction. *Proceedings of the VLDB Endowment*, 6(14):1810–1821, 2013.
24. M. Dylla and M. Theobald. Learning Tuple Probabilities in Probabilistic Databases. Research Report MPI-I-2014-5-001, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, January 2014.
25. M. Dylla, M. Theobald, and I. Miliaraki. Top-k query processing in probabilistic databases with non-materialized views. In *Proceedings of the 29th International Conference on Data Engineering*, ICDE, pages 122–133, Washington, DC, USA, 2013. IEEE Computer Society.
26. R. Fagin, A. Lotem, and M. Naor. Optimal Aggregation Algorithms for Middleware. In *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS, pages 102–113, New York, NY, USA, 2001. ACM.
27. W. Feller. *An introduction to probability theory and its applications*. Wiley, Hoboken, NJ, USA, 3rd edition, 1968.
28. R. Fink and D. Olteanu. On the optimal approximation of queries using tractable propositional languages. In *Proceedings of the 14th International Conference on Database Theory*, ICDT, pages 174–185, New York, NY, USA, 2011. ACM.
29. M. Fisher, D. Gabbay, and L. Vila. *Handbook of Temporal Reasoning in Artificial Intelligence*. Foundations of Artificial Intelligence. Elsevier, Essex, UK, 1st edition, 2005.
30. N. Fuhr and T. Rölleke. A Probabilistic Relational Algebra for the Integration of Information Retrieval and Database Systems. *ACM Transactions on Information Systems*, 15(1):32–66, Jan. 1997.
31. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, NY, USA, 1990.
32. E. Grädel, Y. Gurevich, and C. Hirsch. The Complexity of Query Reliability. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS, pages 227–234, New York, NY, USA, 1998. ACM.
33. T. J. Green and V. Tannen. Models for Incomplete and Probabilistic Information. *IEEE Data Engineering Bulletin*, 29(1):17–24, 2006.
34. J. W. Hector Garcia-Molina, Jeffrey D. Ullman. *Database systems: the complete book*. Prentice-Hall, Upper Saddle River, NJ, USA, 1st edition, 2002.
35. J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. YAGO2: A Spatially and Temporally Enhanced Knowledge Base from Wikipedia. *Artificial Intelligence*, 194:28–61, Jan. 2013.
36. I. F. Ilyas, G. Beskales, and M. A. Soliman. A Survey of Top-k Query Processing Techniques in Relational Database Systems. *ACM Computing Surveys*, 40(4):11:1–11:58, Oct. 2008.
37. C. S. Jensen. *Temporal Database Management*. PhD thesis, Aalborg University, Aalborg, Denmark, April 2000.
38. A. Jha and D. Suciu. Knowledge Compilation Meets Database Theory: Compiling Queries to Decision Diagrams. *Theory of Computing Systems*, 52(3):403–440, Apr. 2013.
39. B. Kanagal and A. Deshpande. Lineage processing over correlated probabilistic databases. In *Proceedings of the International Conference on Management of Data*, SIGMOD, pages 675–686, New York, NY, USA, 2010. ACM.

40. B. Kanagal, J. Li, and A. Deshpande. Sensitivity analysis and explanations for robust query evaluation in probabilistic databases. In *Proceedings of the International Conference on Management of Data*, SIGMOD, pages 841–852, New York, NY, USA, 2011. ACM.
41. P. C. Kanellakis, G. M. Kuper, and P. Z. Revesz. Constraint query languages. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS, pages 299–313, New York, NY, USA, 1990. ACM.
42. S. Khanna, S. Roy, and V. Tannen. Queries with Difference on Probabilistic Databases. *Proceedings of the VLDB Endowment*, 4(11):1051–1062, 2011.
43. C. Koch and D. Olteanu. Conditioning probabilistic databases. *Proceedings of the VLDB Endowment*, 1(1):313–325, Aug. 2008.
44. N. Nakashole, G. Weikum, and F. Suchanek. PATTY: A Taxonomy of Relational Patterns with Semantic Types. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL, pages 1135–1145, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.
45. J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, Berlin, Heidelberg, New York, 2nd edition, 2006.
46. P. Ohrstrom. *Temporal Logic: From Ancient Ideas to Artificial Intelligence*. Springer, Berlin, Heidelberg, New York, 2009.
47. D. Olteanu and J. Huang. Using OBDDs for Efficient Query Evaluation on Probabilistic Databases. In S. Greco and T. Lukasiewicz, editors, *Scalable Uncertainty Management, Second International Conference, SUM 2008*, volume 5291 of *Lecture Notes in Computer Science*, pages 326–340, Berlin, Heidelberg, New York, 2008. Springer.
48. D. Olteanu, J. Huang, and C. Koch. SPROUT: Lazy vs. Eager Query Plans for Tuple-Independent Probabilistic Databases. In *Proceedings of the 25th International Conference on Data Engineering*, ICDE, pages 640–651, Washington, DC, USA, 2009. IEEE Computer Society.
49. D. Olteanu, J. Huang, and C. Koch. Approximate confidence computation in probabilistic databases. In *Proceedings of the 26th International Conference on Data Engineering*, ICDE, pages 145–156, Washington, DC, USA, 2010. IEEE Computer Society.
50. D. Olteanu and H. Wen. Ranking Query Answers in Probabilistic Databases: Complexity and Efficient Algorithms. In *Proceedings of the 28th International Conference on Data Engineering*, ICDE, pages 282–293, Washington, DC, USA, 2012. IEEE Computer Society.
51. C. Ré, N. N. Dalvi, and D. Suciu. Efficient Top-k Query Evaluation on Probabilistic Data. In *Proceedings of the 23rd International Conference on Data Engineering*, ICDE, pages 886–895, Washington, DC, USA, 2007. IEEE Computer Society.
52. C. Ré and D. Suciu. Approximate Lineage for Probabilistic Databases. *Proceedings of the VLDB Endowment*, 1(1):797–808, Aug. 2008.
53. C. Ré and D. Suciu. The trichotomy of HAVING queries on a probabilistic database. *VLDB Journal*, 18(5):1091–1116, 2009.
54. T. Rekatsinas, A. Deshpande, and L. Getoor. Local Structure and Determinism in Probabilistic Databases. In *Proceedings of the International Conference on Management of Data*, SIGMOD, pages 373–384, New York, NY, USA, 2012. ACM.
55. Y. Sagiv and M. Yannakakis. Equivalences Among Relational Expressions with the Union and Difference Operators. *Journal of the ACM*, 27(4):633–655, Oct. 1980.

56. A. D. Sarma, M. Theobald, and J. Widom. Exploiting Lineage for Confidence Computation in Uncertain and Probabilistic Databases. In *Proceedings of the 24th International Conference on Data Engineering*, ICDE, pages 1023–1032, Washington, DC, USA, 2008. IEEE Computer Society.
57. A. D. Sarma, M. Theobald, and J. Widom. LIVE: a lineage-supported versioned DBMS. In *Proceedings of the 22nd international conference on Scientific and statistical database management*, volume 6187 of *Lecture Notes in Computer Science*, pages 416–433, Berlin, Heidelberg, New York, 2010. Springer.
58. P. Sen, A. Deshpande, and L. Getoor. PrDB: managing and exploiting rich correlations in probabilistic databases. *VLDB Journal*, 18(5):1065–1090, Oct. 2009.
59. P. Sen, A. Deshpande, and L. Getoor. Read-once Functions and Query Evaluation in Probabilistic Databases. *Proceedings of the VLDB Endowment*, 3(1-2):1068–1079, Sept. 2010.
60. A. N. Shiryaev. *Probability*. Springer, Berlin, Heidelberg, New York, 2nd edition, 1995.
61. S. Singh, C. Mayfield, S. Mittal, S. Prabhakar, S. E. Hambrusch, and R. Shah. Orion 2.0: native support for uncertain data. In *SIGMOD Conference*, pages 1239–1242, 2008.
62. R. M. Smullyan. *First-order logic*. Springer, Berlin, Heidelberg, New York, 1st edition, 1968.
63. J. Stoyanovich, S. Davidson, T. Milo, and V. Tannen. Deriving Probabilistic Databases with Inference Ensembles. In *Proceedings of the 27th International Conference on Data Engineering*, ICDE, pages 303–314, Washington, DC, USA, 2011. IEEE Computer Society.
64. F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A Core of Semantic Knowledge. In *Proceedings of the 16th International Conference on World Wide Web*, WWW, pages 697–706, New York, NY, USA, 2007. ACM.
65. D. Suciu, D. Olteanu, R. Christopher, and C. Koch. *Probabilistic Databases*. Morgan & Claypool, San Rafael, California, USA, 1st edition, 2011.
66. A. Tuzhilin and J. Clifford. A Temporal Relational Algebra As Basis for Temporal Relational Completeness. In *Proceedings of the 16th International Conference on Very Large Data Bases*, VLDB, pages 13–23, San Rafael, California, USA, 1990. Morgan Kaufmann Publishers.
67. G. A. V. Sperschneider. *Logic: A Foundation for Computer Science*. Addison-Wesley, Boston, MA, USA, 1st edition, 1991.
68. L. G. Valiant. The Complexity of Computing the Permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.
69. J. van Benthem. *The Logic of Time: A Model-Theoretic Investigation into the Varieties of Temporal Ontology and Temporal Discourse*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2nd edition, 1991.
70. G. Weikum and M. Theobald. From Information to Knowledge: Harvesting Entities and Relationships from Web Sources. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS, pages 65–76, New York, NY, USA, 2010. ACM.