Department of Computer Science

INFORMATION INTEGRATION WITH PROVENANCE ON THE SEMANTIC WEB VIA PROBABILISTIC DATALOG+/–

Thomas Lukasiewicz, Maria Vanina Martinez, Livia Predoiu, Gerardo I. Simari

CS-RR-15-01

Department of Computer Science, University of Oxford
Wolfson Building, Parks Road, Oxford, OX1 3QD

DEPARTMENT OF COMPUTER SCIENCE RESEARCH REPORT

DEPARTMENT OF COMPUTER SCIENCE RESEARCH REPORT 14-XX, JANUARY 2015

INFORMATION INTEGRATION WITH PROVENANCE ON THE SEMANTIC WEB VIA PROBABILISTIC DATALOG+/–

31 JANUARY 2015

Thomas Lukasiewicz [1]   Maria Vanina Martinez [2]   Livia Predoiu [3]
Gerardo I. Simari [4]

Abstract. The recently introduced Datalog+/– family of tractable ontology languages is suitable for representing and reasoning over lightweight ontologies, such as $\mathcal{EL}$ and the *DL-Lite* family of description logics. In this paper, we explore the use of Datalog+/– for information integration based on probabilistic data exchange. More specifically, we study the previously introduced probabilistic data exchange problem consisting of a probabilistic database as a source, source-to-target mappings in Datalog+/– and a target Datalog+/– ontology. We provide a complexity analysis for deciding the existence of (deterministic and probabilistic (universal)) solutions in the context of data exchange. In particular, we show that tractability is preserved for simple probabilistic representations, such as tuple-independent ones.

[1]Department of Computer Science, University of Oxford, UK; e-mail: thomas.lukasiewicz@cs.ox.ac.uk.
[2]Departamento de Ciencias e Ingeniería de la Computación, Universidad Nacional del Sur and CONICET, Argentina; e-mail: mvm@cs.uns.edu.ar.
[3]Department of Computer Science, University of Oxford, UK; e-mail: livia.predoiu@cs.ox.ac.uk.
[4]Departamento de Ciencias e Ingeniería de la Computación, Universidad Nacional del Sur and CONICET, Argentina; e-mail: gis@cs.uns.edu.ar.

Acknowledgements: This paper is a significantly extended and revised version of 2 papers that appeared in: *Proceedings of the 9th Workshop on Uncertainty Reasoning for the Semantic Web, (URSW 2013)*, CEUR Proceedings, 2013 [1] and *Uncertainty Reasoning for the Semantic Web III International Workshops URSW 2011-2013, Held at ISWC Revised Selected Papers*, Springer 2014 [2]. This work was supported by the European Research Council under the Marie Curie Fellowship "'PRODIMA"' and by the EPSRC under the grant "PrOQaW".

Contents

1 Introduction

Information integration is widely considered a key (and costly) challenge of our knowledge society [3, 4]. The challenge is complex and includes many interrelated sub-challenges, like identifying which data sources to use when answering a query, creating a common representation for the heterogeneous data sources identified as relevant, extracting data from the sources, cleaning the extracted data, eliminating duplicates by identifying the same objects in different data sources, and transforming the extracted and cleaned data into a unified format.

In the simpler setting of relational information integration, the sources are databases where information is structured with (different) relational database schemas; the sources correspond to so-called *source* or *local* schemas. The unified format is the *target* or *global* schema. Heterogeneity is less challenging, as it is restricted to different kinds of schemas. Unrestricted heterogeneity involves databases hidden behind applications, document repositories, and other kinds of unstructured information. In the Semantic Web, with description logics (DLs) being the underlying knowledge representation formalism, information is structured via ontological schemas. Hence, ontologies replace databases – source databases are replaced by *source* (or *local*) ontologies, and the target database is replaced by a *target* (or *global*) ontology. In the following, we use guarded Datalog+/– as the language for representing the sources, the target, and the mappings – one of the advantages of this language is that it is capable of representing ontological knowledge, but it keeps the notation used in databases for greater readability.

Information integration is usually achieved via mappings between logical formalizations of data sources (cf. [5, 6, 7] for an overview). There are mainly three ways in which information from different sources can be integrated:

- **Data exchange:** Data structured under a source schema S (or different source schemas $S_1, \ldots, S_k$) is transformed into data structured under a different target schema T, and materialized (merged and acquired) there through the mapping.

- **Data integration:** Heterogeneous data in different sources $S_1, \ldots, S_k$ is queried via a virtual global schema T, i.e., no actual exchange of data is needed.

- **Peer-to-peer data integration:** There is no global schema given. All peers $S_1, \ldots,$ S_k are autonomous and independent from each other, and each peer can hold data and be queried. The peers can be viewed as nodes in a network that are linked to other nodes by means of so-called peer-to-peer (P2P) mappings. That is, each source can also be a target for another source.

In this paper, we investigate probabilistic data exchange, which has been proposed for integrating probabilistic databases with either deterministic or probabilistic mappings [8, 9]. We use guarded Datalog+/– [10] as the underlying relational information integration language, which is an ontology language extending plain Datalog by negative constraints and the possibility of rules with existential quantification and equality in rule heads, while restricting the rule syntax by so-called guards in rule bodies to gain decidability and tractability. Essentially, it extends Datalog to negative constraints, tuple-generating dependencies (TGDs), and equality-generating dependencies (EGDs), but suitably restricted to gain decidability and data tractability. In this way, it is possible to capture the *DL-Lite* family of DLs and also the DL $\mathcal{EL}$. As such, guarded Datalog+/– is a very expressive and convenient language for ontology-based database access, which makes it particularly attractive for data exchange on the Semantic Web. For simplicity, from now on, we often refer to "guarded Datalog+/–" simply as "Datalog+/–". Though general data exchange and probabilistic data exchange frameworks often use standard or weakly acyclic sets of TGDs and EGDs for the target, in this work, we adopt linear and guarded TGDs and non-conflicting keys, as proposed in Datalog+/–.

We also sketch how provenance information can be added to Datalog+/– as a mapping language, to be able to track the origin of a mapping for trust assessment and debugging. Capturing the provenance of mappings allows to resolve inconsistencies of mappings by considering the history of their creation. It also helps to detect whether and how to perform mapping updates if the information sources have changed or evolved. Finally, it allows to capture mapping cycles, debug mappings, and to perform meta-reasoning with mappings and the knowledge bases themselves.

This paper extends [1], where we proposed the use of probabilistic Datalog+/– as a language for information integration. Here, we study a theoretical framework for probabilistic data exchange based on Datalog+/– and provide complexity results for deciding the existence of a solution in linear and guarded Datalog+/– for both the deterministic variant and its probabilistic extension. Another difference with respect to [1] is that here, we study data exchange in the presence of a probabilistic source and consider a very general probabilistic model involving unrestricted probability distributions (in [1], we considered a probabilistic extension of Datalog+/– with Markov logic networks).

The rest of this paper is organized as follows. In Section 2, we recall the basics of guarded Datalog+/–. Sections 3 and 4 define our approach to deterministic data exchange on top of Datalog+/– and illustrate the type of ontology mappings that can be expressed in this framework, respectively. In Sections 5 and 6, we generalize to probabilistic data exchange on top of Datalog+/– and the types of probabilistic ontology mappings that it can express, respectively. Section 7 provides complexity results, and Section 8 deals with provenance in our approach. In Sections 9 and 10, we discuss related work, summarize the main results, and give an outlook on future research.

2 Guarded Datalog+/–

We now describe guarded Datalog+/– [10], which here includes negative constraints and (separable) equality-generating dependencies (EGDs). We first describe some preliminaries on databases and queries, and then tuple-generating dependencies (TGDs) and the concept of chase. We finally recall negative constraints and (separable) EGDs, which are other important ingredients of guarded Datalog+/– ontologies.

Databases and Queries. The elementary ingredients are constants, nulls, and variables, which serve as arguments in atomic formulas in databases, queries, and dependencies: (i) a fixed countably infinite universe of *(data) constants* Δ (which constitute the "normal" domain of a database), (ii) a fixed countably infinite set of *(labeled) nulls* Δ_N (used as "fresh" Skolem terms, which are placeholders for unknown values, and can thus be seen as variables), and (iii) a fixed countably infinite set of variables $\mathcal{X}$ (used in queries and dependencies). Different constants represent different values (*unique name assumption*), while different nulls may represent the same value. We assume a lexicographic order on $\Delta \cup \Delta_N$, with every symbol in Δ_N following all symbols in Δ. We denote by $\mathbf{X}$ sequences of variables $X_1, \ldots, X_k$ with $k \geqslant 0$.

We next define atomic formulas, which occur in databases, queries, and dependencies, and which are constructed from relation names and terms, as usual. We assume a *relational schema* $\mathcal{R}$, which is a finite set of *relation names* (or *predicate symbols*, or simply *predicates*). A *position* $P[i]$ identifies the i-th argument of a predicate P. A *term* t is a data constant, null, or variable. An *atomic formula* (or *atom*) $\mathbf{a}$ has the form $P(t_1, \ldots, t_n)$, where P is an n-ary predicate, and $t_1, \ldots, t_n$ are terms. We denote by $\mathit{pred}(\mathbf{a})$ and $\mathit{dom}(\mathbf{a})$ its predicate and the set of all its arguments, respectively. The latter two notations are naturally extended to sets of atoms and conjunctions of atoms. A conjunction of atoms is often identified with the set of all its atoms.

We are now ready to define the notion of a database relative to a relational schema, as well as conjunctive and Boolean conjunctive queries to databases. A *database (instance)* D for a relational schema $\mathcal{R}$ is a (possibly infinite) set of atoms with predicates from $\mathcal{R}$ and arguments from Δ. Such D is *ground* iff it contains only atoms with arguments from Δ. A *conjunctive query (CQ)* over $\mathcal{R}$ has the form $Q(\mathbf{X}) = \exists \mathbf{Y}\, \Phi(\mathbf{X}, \mathbf{Y}, \mathbf{C})$, where $\Phi(\mathbf{X}, \mathbf{Y}, \mathbf{C})$ is a conjunction of atoms with the variables $\mathbf{X}$ and $\mathbf{Y}$, and eventually constants $\mathbf{C}$, but without nulls. Note that $\Phi(\mathbf{X}, \mathbf{Y})$ may also contain equalities but no inequalities. A *Boolean CQ (BCQ)* over $\mathcal{R}$ is a CQ of the form $Q()$. We often write a BCQ as the set of all its atoms, having constants and variables as arguments, and omitting the quantifiers. Answers to CQs and BCQs are defined via *homomorphisms*, which are mappings $\mu\colon \Delta \cup \Delta_N \cup \mathcal{X} \to \Delta \cup \Delta_N \cup \mathcal{X}$ such that (i) $c \in \Delta$ implies $\mu(c) = c$, (ii) $c \in \Delta_N$ implies $\mu(c) \in \Delta \cup \Delta_N$, and (iii) μ is naturally extended to atoms, sets of atoms, and conjunctions of atoms. The set of all *answers* to a CQ $Q(\mathbf{X}) = \exists \mathbf{Y}\, \Phi(\mathbf{X}, \mathbf{Y})$ over a database D, denoted $Q(D)$, is the set of all tuples $\mathbf{t}$ over Δ for which there exists a homomorphism $\mu\colon \mathbf{X} \cup \mathbf{Y} \to \Delta \cup \Delta_N$ such that $\mu(\Phi(\mathbf{X}, \mathbf{Y})) \subseteq D$ and $\mu(\mathbf{X}) = \mathbf{t}$. The *answer* to a BCQ $Q()$ over a database D is *Yes*, denoted $D \models Q$, iff $Q(D) \neq \emptyset$.

Tuple-Generating Dependencies (TGDs). Tuple-generating dependencies (TGDs) describe constraints on databases in the form of generalized Datalog rules with existentially quantified conjunctions of atoms in rule heads; their syntax and semantics are as follows. Given a relational schema $\mathcal{R}$, a *tuple-generating dependency (TGD)* σ is a first-order formula of the form $\forall \mathbf{X} \forall \mathbf{Y}\, \Phi(\mathbf{X}, \mathbf{Y}) \to \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z})$, where $\Phi(\mathbf{X}, \mathbf{Y})$ and $\Psi(\mathbf{X}, \mathbf{Z})$ are conjunctions of atoms over $\mathcal{R}$ called the *body* and the *head* of σ, denoted $\mathit{body}(\sigma)$ and $\mathit{head}(\sigma)$, respectively. A TGD is *guarded* iff it contains an atom in its body that involves all variables appearing in the body. The leftmost such atom is the *guard atom* (or *guard*) of σ. The non-guard atoms in the body of σ are the side atoms of σ. We usually omit the universal quantifiers in TGDs. Such σ is satisfied in a database D for $\mathcal{R}$ iff, whenever there exists a homomorphism h that maps the atoms of $\Phi(\mathbf{X}, \mathbf{Y})$ to atoms of D, there exists an extension h' of h that maps the atoms of $\Psi(\mathbf{X}, \mathbf{Z})$ to atoms of D. All sets of TGDs are finite here.

Query answering under TGDs, i.e., the evaluation of CQs and BCQs on databases under a set of TGDs is defined as follows. For a database D for $\mathcal{R}$, and a set of TGDs Σ on $\mathcal{R}$, the set of *models* of D and Σ, denoted $\mathit{mods}(D, \Sigma)$, is the set of all (possibly infinite) databases B such that (i) $D \subseteq B$ (ii) every $\sigma \in \Sigma$ is satisfied in B. The set of *answers* for a CQ Q to D and Σ, denoted $\mathit{ans}(Q, D, \Sigma)$, is the set of all tuples $\mathbf{a}$ such that $\mathbf{a} \in Q(B)$ for all $B \in \mathit{mods}(D, \Sigma)$. The *answer* for a BCQ Q to D and Σ is *Yes*, denoted $D \cup \Sigma \models Q$, iff $\mathit{ans}(Q, D, \Sigma) \neq \emptyset$. We recall that query answering under TGDs is equivalent to query answering under TGDs with only single atoms in their heads. We thus often assume w.l.o.g. that every TGD has a single atom in its head.

The Chase. The *chase* was introduced to enable checking implication of dependencies [11] and later also for checking query containment [12]. It is a procedure for repairing a database relative to a set of dependencies, so that the result of the chase satisfies the dependencies. By "chase", we refer both to the chase procedure and to its output. The TGD chase works on a database through so-called TGD *chase rules* (an extended chase with also equality-generating dependencies is discussed below). The TGD chase rule comes in two flavors: *restricted* and *oblivious*, where the restricted one applies TGDs only when they are not satisfied (to repair them), while the oblivious one always applies TGDs (if they produce a new result). We focus on the oblivious one here; the *(oblivious) TGD chase rule* defined below is the building block of the chase.

TGD CHASE RULE. Consider a database D for a relational schema $\mathcal{R}$, and a TGD σ on $\mathcal{R}$ of the form $\Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z})$. Then, σ is *applicable* to D if there exists a homomorphism h that maps the atoms of $\Phi(\mathbf{X}, \mathbf{Y})$ to atoms of D. Let σ be applicable to D, and h_1 be a homomorphism that extends h as follows: for each $X_i \in \mathbf{X}$, $h_1(X_i) = h(X_i)$; for each $Z_j \in \mathbf{Z}$, $h_1(Z_j) = z_j$, where z_j is a "fresh" null, i.e., $z_j \in \Delta_N$, z_j does not occur in D, and z_j lexicographically follows all other nulls already introduced. The *application of* σ on D adds to D the atom $h_1(\Psi(\mathbf{X}, \mathbf{Z}))$ if not already in D. ■

The chase algorithm for a database D and a set of TGDs Σ consists of an exhaustive application of the TGD chase rule in a breadth-first (level-saturating) fashion, which leads as result to a (possibly infinite) chase for D and Σ. Formally, the *chase of level up to* 0 of D relative to Σ, denoted $\mathit{chase}^0(D, \Sigma)$, is defined as D, assigning to every atom in D the *(derivation) level* 0. For every $k \geqslant 1$, the *chase of level up to k* of D relative to Σ, denoted $\mathit{chase}^k(D, \Sigma)$, is constructed as follows: let $I_1, \ldots, I_n$ be all possible images of bodies of TGDs in Σ relative to some homomorphism such that (i) $I_1, \ldots, I_n \subseteq \mathit{chase}^{k-1}(D, \Sigma)$ and (ii) the highest level of an atom in every I_i is $k-1$; then, perform every corresponding TGD application on $\mathit{chase}^{k-1}(D, \Sigma)$, choosing the applied TGDs and homomorphisms in a (fixed) linear and lexicographic order, respectively, and assigning to every new atom the *(derivation) level k*. The *chase* of D relative to Σ, denoted $\mathit{chase}(D, \Sigma)$, is then defined as the limit of $\mathit{chase}^k(D, \Sigma)$ for $k \rightarrow \infty$.

The (possibly infinite) chase relative to TGDs is a *universal model*, i.e., there exists a homomorphism from $\mathit{chase}(D, \Sigma)$ onto every $B \in \mathit{mods}(D, \Sigma)$ [13, 14]. This result implies that BCQs Q over D and Σ can be evaluated on the chase for D and Σ, i.e., $D \cup \Sigma \models Q$ is equivalent to $\mathit{chase}(D, \Sigma) \models Q$. In the case of guarded TGDs Σ, such BCQs Q can be evaluated on an initial fragment of $\mathit{chase}(D, \Sigma) \models Q$ of constant depth $k \cdot |Q|$, and thus be done in polynomial time in the data complexity.

Note that sets of guarded TGDs (with single-atom heads) are theories in the guarded fragment of first-order logic [15]. Note also that guardedness is a truly fundamental class ensuring decidability as adding a single unguarded Datalog rule to a guarded Datalog+/– program may destroy decidability as shown in [13].

Negative Constraints. Another crucial ingredient of Datalog+/– for ontological modeling are *negative constraints (NCs*, or simply *constraints*), which are first-order formulas of the form $\forall \mathbf{X}\ \Phi(\mathbf{X}) \rightarrow \bot$, where $\Phi(\mathbf{X})$ is a conjunction of atoms (not necessarily guarded). We usually omit the universal quantifiers, and we implicitly assume that all sets of constraints are finite here. Adding negative constraints to answering BCQs Q over databases and guarded TGDs is computationally easy, as for each constraint $\forall \mathbf{X} \Phi(\mathbf{X}) \rightarrow \bot$, we only have to check that the BCQ $\Phi(\mathbf{X})$ evaluates to false; if one of these checks fails, then the answer to the original BCQ Q is true, otherwise the negative constraints can be simply ignored when answering the original BCQ Q.

Equality-Generating Dependencies (EGDs). A further important ingredient of Datalog+/– for modeling ontologies are *equality-generating dependencies* (or *EGDs*) σ, which are first-order formulas $\forall \mathbf{X}\, \Phi(\mathbf{X}) \rightarrow X_i = X_j$, where $\Phi(\mathbf{X})$, called the *body* of σ, denoted $\mathit{body}(\sigma)$, is a (not necessarily guarded) conjunction of atoms, and X_i and X_j are variables from $\mathbf{X}$. We call $X_i = X_j$ the *head* of σ, denoted $\mathit{head}(\sigma)$. Such σ is satisfied in a database D for $\mathcal{R}$ iff, whenever there exists a homomorphism h such that $h(\Phi(\mathbf{X}, \mathbf{Y})) \subseteq D$, it holds that $h(X_i) = h(X_j)$. We usually omit the universal quantifiers in EGDs, and all sets of EGDs are finite here.

An EGD σ on $\mathcal{R}$ of the form $\Phi(\mathbf{X}) \rightarrow X_i = X_j$ is *applicable* to a database D for $\mathcal{R}$ iff there exists a homomorphism $\eta\colon \Phi(\mathbf{X}) \rightarrow D$ such that $\eta(X_i)$ and $\eta(X_j)$ are different and not both constants. If $\eta(X_i)$ and $\eta(X_j)$ are different constants in Δ, then there is a *hard violation* of σ (and, as we will see below, the *chase* fails). Otherwise, the result of the application of σ to D is the database $h(D)$ obtained from D by replacing every occurrence of a non-constant element $e \in \{\eta(X_i), \eta(X_j)\}$ in D by the other element e' (if e and e' are both nulls, then e precedes e' in the lexicographic order). The *chase* of a database D, in the presence of two sets Σ_T and Σ_E of TGDs and EGDs, respectively, denoted $\mathit{chase}(D, \Sigma_T \cup \Sigma_E)$, is computed by iteratively applying (1) a single TGD once, according to the standard order and (2) the EGDs, as long as they are applicable (i.e., until a fixpoint is reached). To assure that adding EGDs to answering BCQs Q over databases and guarded TGDs along with negative constraints does not increase the complexity of query answering, all EGDs are assumed to be *separable* [10] (one such class of separable EGDs are non-conflicting keys [10]). Intuitively, separability holds whenever: (i) if there is a hard violation of an EGD in the chase, then there is also one on the database w.r.t. the set of EGDs alone (i.e., without considering the TGDs); and (ii) if there is no chase failure, then the answers to a BCQ w.r.t. the entire set of dependencies equals those w.r.t. the TGDs alone (i.e., without the EGDs).

Guarded Datalog+/– Ontologies. We define (guarded) Datalog+/– ontologies as follows. A *(guarded) Datalog+/– ontology* consists of a database D, a (finite) set of guarded TGDs Σ_T, a (finite) set of negative constraints Σ_C, and a (finite) set of EGDs Σ_E that are separable from Σ_T.

3 Deterministic Data Exchange

In this section, we recall the classical logical framework of data exchange and data integration [6, 9] in the context of both deterministic and probabilistic databases, and tailor the framework to suit data exchange and data integration between Datalog+/– ontologies. The syntax of a schema mapping in Datalog+/– is defined as follows.

Definition 1 (Schema Mapping)**.** A *schema mapping* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ consists of a source schema $\mathbf{S} = \{S_1, \ldots, S_n\}$, a target schema $\mathbf{T} = \{T_1, \ldots, T_m\}$ disjoint from $\mathbf{S}$, and a set $\Sigma = \Sigma_{st} \cup \Sigma_t$ of TGDs, negative constraints, and non-conflicting keys, where Σ_{st} are *source-to-target* TGDs, negative constraints, and non-conflicting keys over $\mathbf{S} \cup \mathbf{T}$, and Σ_t are *target* TGDs, negative constraints, and non-conflicting keys over $\mathbf{T}$.

The semantics of schema mappings is defined by relating source and target databases in a semantically meaningful and consistent way. More specifically, in the case of data exchange between deterministic databases, a target database J over Δ is considered to be a *solution* of the source database I over Δ for the data exchange problem specified via the schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ iff $I \cup J \models \Sigma$.

Definition 2 (Solution)**.** A target database J over Δ is a *solution* for a source database I over Δ relative to a schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ iff $I \cup J \models \Sigma$. We denote by $\mathit{Sol}_{\mathcal{M}}$ the set of all pairs (I, J) of source databases I and target databases J with $I \cup J \models \Sigma$.

There are many possible solutions J to a source database I relative to $\mathcal{M}$ in $\mathit{Sol}_{\mathcal{M}}$. Among all such solutions, the preferred solutions are the ones that carry only the necessary information for data exchange; i.e., all the constants of the source database that can be transferred via the mapping are included in the target database. Such solutions are called *universal* solutions. Similar to universal models in the context of the chase derivation of Datalog+/– (see Section 2), a universal solution can be homomorphically mapped to all other solutions leaving the constants unchanged.

Definition 3 (Universal Solution)**.** A target database J over Δ is a *universal solution* for a source database I over Δ relative to a schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ iff (i) J is a solution, and (ii) for each solution

J' for I relative to $\mathcal{M}$, there is a homomorphism $h\colon J \to J'$. We denote by $\mathit{USol}_{\mathcal{M}}$ ($\subseteq \mathit{Sol}_{\mathcal{M}}$) the set of all pairs (I, J) of source databases I and target databases J such that J is a universal solution for I relative to $\mathcal{M}$.

For defining the data exchange problem over probabilistic databases, we first need to define probabilistic databases, as they serve as source and target databases in the probabilistic data exchange setting.

Definition 4 (Probabilistic Database)**.** A *probabilistic database over* $\mathcal{R}$ is a probability space $Pr = (\mathcal{I}, \mu)$ such that $\mathcal{I}$ is the set of all (possibly infinitely many) standard databases over $\mathcal{R}$, and $\mu\colon \mathcal{I} \to [0,1]$ is a function that satisfies $\sum_{I\in\mathcal{I}} \mu(I) = 1$.

In this paper, we adopt a compact encoding of probabilistic databases by annotating database atoms with Boolean combinations of elementary events, where every annotation describes when the atom is true and is associated with a probability. We first define annotations and annotated atoms.

Definition 5 (Annotations and Annotated Atoms)**.** Let $e_1, \ldots, e_n$ be $n \geqslant 1$ *elementary events*. A *world* w is a conjunction $\ell_1 \wedge \cdots \wedge \ell_n$, where each ℓ_i, $i \in \{1, \ldots, n\}$, is either the elementary event e_i or its negation $\neg e_i$. An *annotation* λ is any Boolean combination of elementary events (i.e., all elementary events are annotations, and if λ_1 and λ_2 are annotations, then also $\neg\lambda_1$ and $\lambda_1 \wedge \lambda_2$); as usual, $\lambda_1 \vee \lambda_2$ abbreviates $\neg(\neg\lambda_1 \wedge \neg\lambda_2)$. An *annotated atom* has the form $a\colon \lambda$, where a is an atom, and λ is an annotation.

Based on this definition, we can now define the compact encoding of probabilistic databases that we will be using.

Definition 6 (Compact Encoding of Probabilistic Databases)**.** A set D of annotated atoms over Δ along with a probability $\mu(w) \in [0,1]$ for every world w *compactly encodes a probabilistic database* by defining:

1. the probability of every annotation λ as the sum of the probabilities of all worlds in which λ is true, and

2. the probability of every database $\{a_1, \ldots, a_m\}$ such that $\{a_1\colon \lambda_1, \ldots, a_m\colon \lambda_m\} \subseteq D$ for some annotations $\lambda_1, \ldots, \lambda_m$ as the probability of $\lambda_1 \wedge \cdots \wedge \lambda_m$ (and the probability of every other database as 0).

While the syntax of a deterministic schema mapping over probabilistic databases does not change, its semantics needs to be adapted. Exchanging data between probabilistic databases means that a properly defined joint probability space *Pr* over the solution relation $\mathit{Sol}_{\mathcal{M}}$ and the universal solution relation $\mathit{USol}_{\mathcal{M}}$ must exist. Note that $\mathit{Sol}_{\mathcal{M}}$ and $\mathit{USol}_{\mathcal{M}}$ exist in probabilistic data exchange as well, because joint events (I, J) that can be constructed for pairs of probabilistic source instances $Pr_s = (\mathcal{I}, \mu_s)$ and probabilistic target instances $Pr_t = (\mathcal{J}, \mu_t)$ need also to satisfy the condition $I \cup J \models \Sigma$ to be considered as semantic components of probabilistic solutions.

As stated more formally below, a properly defined joint probability distribution *Pr* over the solution relation $\mathit{Sol}_{\mathcal{M}}$ or the universal solution relation $\mathit{USol}_{\mathcal{M}}$ requires to match each of the given marginal distributions Pr_s and Pr_t. These constraints over the marginal distributions of *Pr* are called $\mathit{Sol}_{\mathcal{M}}$-match and $\mathit{USol}_{\mathcal{M}}$-match [9], as they are defined over $\mathit{Sol}_{\mathcal{M}}$ and $\mathit{USol}_{\mathcal{M}}$, respectively.

Definition 7 (Probabilistic (Universal) Solution)**.** A probabilistic target database $Pr_t = (\mathcal{J}, \mu_t)$ is a *probabilistic solution* (resp., *probabilistic universal solution*) for a probabilistic source database $Pr_s = (\mathcal{I}, \mu_s)$ relative to a schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ iff there exists a probabilistic space $Pr = (\mathcal{I} \times \mathcal{J}, \mu)$ that satisfies the following two conditions:

1. The left and right marginals of *Pr* are Pr_s and Pr_t, respectively. That is,
   (a) $\sum_{J \in \mathcal{J}}(\mu(I, J)) = \mu_s(I)$ for all $I \in \mathcal{I}$ and
   (b) $\sum_{I \in \mathcal{I}}(\mu(I, J)) = \mu_t(J)$ for all $J \in \mathcal{J}$;
2. $\mu(I, J) = 0$ for all $(I, J) \notin \mathit{Sol}_{\mathcal{M}}$ (resp., $(I, J) \notin \mathit{USol}_{\mathcal{M}}$).

For mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ and query $Q(\mathbf{X}) = \exists \mathbf{Y}\, \Phi(\mathbf{X}, \mathbf{Y}, \mathbf{C})$ (as introduced in Section 2), query answering within the data exchange setting of a source database I is defined as deriving the certain answers, i.e., the tuples consisting of constants that belong to $Q(J)$ for all solutions J for I relative to $\mathcal{M}$. In the probabilistic generalization, each probabilistic target database defines a probability with which a tuple of constants belongs to $Q(J)$ (which is the sum of the probabilities of all standard target databases in which the query evaluates to true), and the probability that this tuple belongs to the answer of the query is the infimum of all such probabilities. In the following definition, we also generalize queries to unions of conjunctive queries (UCQs).

Definition 8 (UCQs)**.** A *union of conjunctive queries* (or *UCQ*) has the form $Q(\mathbf{X}) = \bigvee_{i=1}^{k} \exists \mathbf{Y}_i\ \Phi_i(\mathbf{X}, \mathbf{Y}_i, \mathbf{C}_i)$, where each $\exists \mathbf{Y}_i\, \Phi_i(\mathbf{X}, \mathbf{Y}_i, \mathbf{C}_i)$ with $i \in \{1, \ldots, k\}$ is a CQ with exactly the variables $\mathbf{X}$ and $\mathbf{Y}_i$, and the constants $\mathbf{C}_i$. Given a schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, a probabilistic source database $Pr_s = (\mathcal{I}, \mu_s)$, a UCQ $Q(\mathbf{X}) = \bigvee_{i=1}^{k} \exists \mathbf{Y}_i\, \Phi_i(\mathbf{X}, \mathbf{Y}_i, \mathbf{C}_i)$, and a tuple $\mathbf{A}$ (being a ground instance of $\mathbf{X}$ in Q) over Δ, the *confidence* of $\mathbf{A}$ relative to Q, denoted $\mathit{conf}_Q(\mathbf{A})$, in Pr_s relative to $\mathcal{M}$ is the infimum of $Pr_t(Q(\mathbf{A}))$ subject to all probabilistic solutions Pr_t for Pr_s relative to $\mathcal{M}$. Here, $Pr_t(Q(\mathbf{A}))$ for $Pr_t = (\mathcal{J}, \mu_t)$ is the sum of all $\mu_t(J)$ such that $Q(\mathbf{A})$ evaluates to true in the database $J \in \mathcal{J}$ (i.e., some BCQ $\exists \mathbf{Y}_i\, \Phi_i(\mathbf{A}, \mathbf{Y}_i, \mathbf{C}_i)$ with $i \in \{1, \ldots, k\}$ evaluates to true in J).

The following are the main computational tasks that we consider in this paper.

Existence of a solution (resp., universal solution): Given a schema mapping $\mathcal{M}$ and a probabilistic source database Pr_s, decide whether there exists a probabilistic (resp., probabilistic universal) solution for Pr_s relative to $\mathcal{M}$.

Materialization of a solution (resp., universal solution): Given a schema mapping $\mathcal{M}$ and a probabilistic source database Pr_s, compute a probabilistic solution (resp., probabilistic universal) solution for Pr_s relative to $\mathcal{M}$, if it exists.

Answering UCQs: Given a schema mapping $\mathcal{M}$, a probabilistic source database Pr_s, a UCQ $Q(\mathbf{X})$, and a tuple $\mathbf{A}$ over Δ, compute $\mathit{conf}_Q(\mathbf{A})$ in Pr_s relative to $\mathcal{M}$.

4 Ontology Mappings with Datalog+/–

Mapping languages are formal knowledge representation languages that are chosen according to specific criteria. The two most important criteria are the *expressive power* needed for specifying desired data interoperability tasks on the one hand and the *tractability* of dealing with that language, i.e., query answering, checking for solutions, materializing solutions, etc., on the other. It is well known that there is a tradeoff between expressivity and tractability [16] – the latter is often attained via algorithmic properties that imply certain structural properties, such as the existence of universal solutions after performing a bounded number of computations.

In the following, we examine Datalog+/– as a mapping language. As a language lying in the intersection of the DL and the logic programming paradigms, Datalog+/– allows to integrate the information available in ontologies and, hence, nicely ties together the results on data exchange and integration in databases and the work on ontology mediation in the Semantic Web.

When integrating ontologies with Datalog+/– via *source-to-target TGDs* (for short, we often refer to them as s-t TGDs), such TGDs correspond to *GLAV (global-local-as-view) dependencies* and are used as mappings. In their most general form, TGDs are (as mentioned above) first-order formulas $\forall \mathbf{X}\, \phi(\mathbf{X}) \rightarrow \exists \mathbf{Y} \psi(\mathbf{X}, \mathbf{Y})$ with $\mathbf{X}$ and $\mathbf{Y}$ being tuples of variables, $\phi(\mathbf{X})$ and $\psi(\mathbf{X}, \mathbf{Y})$ being a conjunction of atomic formulas.

The following two types of dependencies are important special cases of source-to-target TGDs: LAV (local as view) and GAV (global as view):

- A **LAV (local as view)** dependency is a source-to-target TGD with a single atom in the body, i.e., of the form $\forall \mathbf{X}\; A_S(\mathbf{X}) \rightarrow \exists \mathbf{Y} \psi(\mathbf{X}, \mathbf{Y})$, where A_S is an atom over the source schema, and $\psi(\mathbf{X}, \mathbf{Y})$ is a conjunction of atoms over the target schema.

- A **GAV (global as view)** dependency is a source-to-target TGD with a single atom in the head, i.e., of the form $\forall \mathbf{X}\; \phi(\mathbf{X}) \rightarrow A_T(\mathbf{X}')$, where $\phi(\mathbf{X})$ is a conjunction of atoms over the source schema, and $A_T(\mathbf{X}')$ is an atom over the target schema with $\mathbf{X}' \subseteq \mathbf{X}$.

The following mappings that are mentioned in [17] as "essential" can also be represented in Datalog+/– (all examples below stem from a consideration of the OAEI benchmark set; more specifically, ontologies 101 and 301–303):

- **Copy (Nicknaming):** Copy a source relation (or concept or role) (of arbitrary arity n) into a target relation (or concept or role) (of the same arity n like the source relation (or concept or role)) and rename it. Note that this kind of mapping is a LAV and a GAV mapping at the same time. For example:

  $$\forall x, y \quad S : \textit{location}(x, y) \rightarrow \quad T : \textit{address}(x, y).$$

- **Projection (Column Deletion):** Create a target relation (or concept or role) by deleting one or more columns of a source relation (or concept or role) (of arbitrary arity $n \geq 2$). Note that this kind of mapping is a LAV and GAV mapping at the same time. For instance:

  $$\forall x, y \quad S : \textit{author}(x, y) \rightarrow \quad T : \textit{person}(x).$$

- **Augmentation (Column Addition):** Create a target relation (or concept or role) (of arbitrary arity $n \geq 2$) by adding one or more columns to the source relation (or concept or role). Note that this is a LAV dependency. A simple example follows:

  $$\forall x \quad S : \textit{editor}(x) \rightarrow \exists z \quad T : \textit{hasEditor}(z, x).$$

- **Decomposition:** Decompose a source relation (or concept or role) (of arbitrary arity n) into two or more target relations (or concepts or roles). Note that this is a LAV dependency. For instance, we can have:

$$\forall x, y \quad S: \textit{publisher}(x, y) \rightarrow \quad T: \textit{organization}(x), \quad T: \textit{proceedings}(y).$$

Only one mapping construct mentioned in [17] as essential – the join – cannot be represented by guarded Datalog+/–. As each TGD has to be guarded, there must be an atom in the body that contains all non-existentially quantified variables and, hence, a join like $\forall x, y \quad S: \textit{book}(y), \ S: \textit{person}(x) \rightarrow \ T: \textit{author}(x, y)$ cannot be represented in guarded Datalog+/–. This, however, can also be considered as a benefit, as joins usually need more computing resources, because they require a large number of operations. Note that the join is introduced for query answering by means of conjunctive queries that we are using to query the target database. This is equivalent to the join in databases.

In ontology mediation, a *mapping* or *alignment* is based on correspondences between so-called matchable entities of two ontologies. The following definition is based on [18]; let S and T be two ontologies (the source and the target) that are to be mapped onto each other, and let q be a function that defines the sets of matchable entities $q(S)$ and $q(T)$. Then, a correspondence between S and T is a triple $\langle e_1, e_2, r\rangle$ with $e_1 \in q(S)$, $e_2 \in q(T)$, and r being an alignment relation between the two matchable elements (note that equivalence and implication are examples of such an alignment relation if the chosen mapping language supports them). A *mapping* or *alignment* between S and T is then a set of correspondences $C = \cup_{i,j,k}\{\langle e_i, e_j, r_k\rangle\}$ between S and T. This is a very general definition, which allows to describe many types of mappings.

Definition 9 (Ontology Mapping [18])**.** Let S be a source ontology, and T be a target ontology. Let q be a function that defines the sets of matchable entities $q(S)$ and $q(T)$. Then, a *correspondence* between S and T is a triple $\langle e_1, e_2, r\rangle$, where $e_1 \in q(S)$, $e_2 \in q(T)$, and r is an alignment relation between the two matchable elements e_1 and e_2. An *ontology mapping* between S and T is a set of correspondences $C = \cup_{i,j,k}\{\langle e_i, e_j, r_k\rangle\}$ between S and T.

Semantic Web and ontology mapping languages usually contain a subset of the aforementioned mapping expressions, plus additional mapping expressions in the form of constraints, which usually are used to specify class disjointness (see, e.g., [7, 19]). However, note that both the data exchange and the ontology mediation communities have also proposed mapping languages that are more expressive than source-to-target TGDs, consisting of full general Datalog expressions and also containing existentially quantified variables in the head – e.g., second-order mappings as described in the requirements of [20] or second-order TGDs [21]. Of course, such mapping languages have less desirable tractability properties. In [19], a probabilistic mapping language is presented that is based on Markov logic networks, built by mappings of basic DL axioms onto predicates with the desired semantics. A closer look reveals that the deterministic mapping constructs that are used are renaming, decomposition, and class disjointness constraints, as well as their combinations. Such disjointness constraints can be modeled with Datalog+/–, using negative constraints (NCs), such as:

- **Disjointness of ontology entities with the same arity:** A source relation (or concept or role) with arity n is disjoint from another relation (or concept or role) with the same arity n. The NC below corresponds to class disjointness that specifies that persons cannot be addresses:

$$\forall x \quad S: \textit{Person}(x), \ T: \textit{Address}(x) \rightarrow \bot.$$

- **Disjointness of ontology entities with different arity:** A source relation (or concept or role) with arity $n \geq 2$ is disjoint from another relation (or concept or role) with the arity $n > m \geq 1$. The example below specifies that persons cannot be bought and, hence, do not have prices.

$$\forall x, y \quad S : \textit{Person}(x),\ \ T : \textit{hasPrice}(x, y) \rightarrow \bot.$$

EGDs are also part of some mapping languages, especially in the database area, and can be represented by Datalog+/– as long as they are separable from the TGDs. Such kinds of dependencies allow to create mappings like the following one specifying that publishers of the same book or journal, in both the source and target schema (or ontology), have to be the same:

$$\forall x, y, z \quad S : \textit{publisher}(x, y),\ \ T : \textit{publishes}(y, z) \rightarrow x = z.$$

5 Probabilistic Data Exchange

Probabilistic data exchange extends classical data exchange by the demand of two database instances I and J not only meeting the deterministic constraints of solutions, but also the probabilities specified by a probability distribution over a set of deterministic schema mappings, which is expressed in the following definition of probabilistic schema mappings.

Definition 10 (Probabilistic Schema Mapping)**.** A *probabilistic schema mapping* is a tuple of the form $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma, \mu)$, consisting of a source schema $\mathbf{S} = \{S_1, \ldots, S_n\}$, a target schema $\mathbf{T} = \{T_1, \ldots, T_m\}$ disjoint from $\mathbf{S}$, a set $\Sigma = \Sigma_{st} \cup \Sigma_t$ of TGDs, negative constraints, and non-conflicting keys, where Σ_{st} are *source-to-target* TGDs, negative constraints, and non-conflicting keys over $\mathbf{S} \cup \mathbf{T}$, and Σ_t are *target* TGDs, negative constraints, and non-conflicting keys over $\mathbf{T}$, and a function $\mu\colon 2^{\Sigma}_{st} \rightarrow [0, 1]$ such that $\sum_{\Sigma' \subseteq \Sigma_{st}} \mu(\Sigma') = 1$.

Probabilistic schema mappings are compactly encoded in the same way as probabilistic databases by annotating TGDs, negative constraints, and non-conflicting keys with Boolean combinations of elementary events. If such mappings are given along with probabilistic databases, then both are compactly encoded via two (not necessarily disjoint) sets of elementary events, assuming that the probabilities of common elementary events are the same in both compact encodings. Note that in probabilistic Datalog+/– [22], which we used for the probabilistic mappings in [1], both sets of elementary events coincide, the annotations are conjunctions of elementary events, and the probability of every world is defined via a Markov logic network.

The next definition lifts the notion of probabilistic (universal) solution from probabilistic source databases under deterministic schema mappings to probabilistic source databases under probabilistic schema mappings.

Definition 11 (Probabilistic (Universal) Solution)**.** A probabilistic target database $\textit{Pr}_t = (\mathcal{J}, \mu_t)$ is a *probabilistic solution* (resp., *probabilistic universal solution*) for a probabilistic source database $\textit{Pr}_s = (\mathcal{I}, \mu_s)$ relative to a probabilistic schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma, \mu_m)$ iff there exists a probabilistic space $\textit{Pr} = (\mathcal{I} \times \mathcal{J} \times 2^{\Sigma}, \mu)$ that satisfies the following two conditions:

1. The three marginals of μ are μ_s, μ_t, and μ_m, such that:
   (a) $\sum_{J \in \mathcal{J},\, \Sigma' \subseteq \Sigma} \mu(I, J, \Sigma') = \mu_s(I)$ for all $I \in \mathcal{I}$,
   (b) $\sum_{I \in \mathcal{I},\, \Sigma' \subseteq \Sigma} \mu(I, J, \Sigma') = \mu_t(J)$ for all $J \in \mathcal{J}$, and

(c) $\sum_{I \in \mathcal{I},\, J \in \mathcal{J}} \mu(I, J, \Sigma') = \mu_m(\Sigma')$ for all $\Sigma' \subseteq \Sigma$;

2. $\mu(I, J, \Sigma') = 0$ for all $(I, J) \notin \mathit{Sol}_{(\mathbf{S},\mathbf{T},\Sigma')}$ (resp., $(I, J) \notin \mathit{USol}_{(\mathbf{S},\mathbf{T},\Sigma')}$).

Using the above probabilistic and probabilistic universal solutions for probabilistic source databases under probabilistic schema mappings, the semantics of UCQs can easily be lifted from deterministic to probabilistic schema mappings as follows.

Definition 12 (UCQs)**.** A *union of conjunctive queries* (or *UCQ*) has the form $Q(\mathbf{X}) = \bigvee_{i=1}^{k} \exists \mathbf{Y}_i \;\Phi_i(\mathbf{X}, \mathbf{Y}_i, \mathbf{C}_i)$, where each $\exists \mathbf{Y}_i \,\Phi_i(\mathbf{X}, \mathbf{Y}_i, \mathbf{C}_i)$ with $i \in \{1, \ldots, k\}$ is a CQ with exactly the variables $\mathbf{X}$ and $\mathbf{Y}_i$, and the constants $\mathbf{C}_i$. Given a probabilistic schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma, \mu_m)$, a probabilistic source database $Pr_s = (\mathcal{I}, \mu_s)$, a UCQ $Q(\mathbf{X}) = \bigvee_{i=1}^{k} \exists \mathbf{Y}_i \,\Phi_i(\mathbf{X}, \mathbf{Y}_i, \mathbf{C}_i)$, and a tuple $\mathbf{A}$ over Δ, the *confidence* of $\mathbf{A}$ relative to Q, denoted $\mathit{conf}_Q(\mathbf{A})$, in Pr_s relative to $\mathcal{M}$ is the infimum of $Pr_t(Q(\mathbf{A}))$ subject to all probabilistic solutions Pr_t for Pr_s relative to $\mathcal{M}$. Here, $Pr_t(Q(\mathbf{A}))$ for $Pr_t = (\mathcal{J}, \mu_t)$ is the sum of all $\mu_t(J)$ such that $Q(\mathbf{A})$ evaluates to true in the database $J \in \mathcal{J}$ (i.e., some BCQ $\exists \mathbf{Y}_i \,\Phi_i(\mathbf{A}, \mathbf{Y}_i, \mathbf{C}_i)$ with $i \in \{1, \ldots, k\}$ evaluates to true in J).

Similarly, the main computational tasks of this paper can easily be generalized from deterministic to probabilistic schema mappings as follows.

Existence of a solution (resp., universal solution): Given a probabilistic schema mapping $\mathcal{M}$ and a probabilistic source database Pr_s, decide whether there exists a probabilistic (resp., probabilistic universal) solution for Pr_s relative to $\mathcal{M}$.

Materialization of a solution (resp., universal solution): Given a probabilistic schema mapping $\mathcal{M}$ and a probabilistic source database Pr_s, compute a probabilistic solution (resp., probabilistic universal) solution for Pr_s relative to $\mathcal{M}$, if it exists.

Answering UCQs: Given a probabilistic schema mapping $\mathcal{M}$, a probabilistic source database Pr_s, a UCQ $Q(\mathbf{X})$, and a tuple $\mathbf{A}$ over Δ, compute $\mathit{conf}_Q(\mathbf{A})$ in Pr_s relative to $\mathcal{M}$.

6 Ontology Mappings with Probabilistic Datalog+/–

In general, probabilistic mappings $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma, \mu)$ in probabilistic Datalog+/– have a similar expressivity as deterministic mappings $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ in Datalog+/–; that is, they can encode LAV and GAV mappings and also the mappings mentioned in Section 4 as being essential like *Copy (Nicknaming), Projection (Column Deletion), Augmentation (Column Addition), Decomposition*, and *Disjointness constraints* of entities with same or different arity (see Section 4 for details).

In addition, in probabilistic Datalog+/–, the above-mentioned kinds of deterministic mappings $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ are extended with uncertainty by defining a probability space $\Omega(2^{\Sigma}, \mu_m)$ over the dependencies Σ in Datalog+/– such that each mapping m_i holds with a probability $\mu_m(m_i)$. The mappings together with the probability space defined over them is represented by the compact encoding defined in Section 3 for probabilistic databases and used in Section 5 for probabilistic mappings as well. This compact encoding of probabilistic databases and mappings considers conditions or events under which the mappings are true or not. These events are not part of the databases, but can represent databases or other relevant events. As such, the two tasks of database and mapping dependencies modeling are separated from the task of modeling the uncertainty around the axioms of the ontology. Note that the set of events used to encode the probabilistic mappings can overlap with the set of events used to encode the probabilistic databases, and their probability can depend completely or in part on the same events.

As an example, consider a tuple-independent (see Section 7) database and a mapping consisting of a single rule. Without loss of generality, let Σ_{st} consist of only one mapping dependency – e.g., the first one mentioned in Section 4 (Copy/Nicknaming):

$$\forall x, y \quad S : \mathit{location}(x, y) \rightarrow \quad T : \mathit{address}(x, y).$$

The probabilistic version of this mapping for a probabilistic source database with only two possible probabilistic tuples *(location(CS Department, Wolfson Building Oxford), 0.8), (location(Maths Department, Andrew Wiles Building Oxford), 0.9)*, which are independent, has probability 0.98 of being true. In this case, clearly, the probability of the mapping is dependent on the probabilistic source database.

Another, more expressive, possibility to encode the probabilistic events under which the mappings hold is to use Markov logic networks (MLNs) as done in [22] – in [1], we suggested the use of the probabilistic extension of Datalog+/– presented in [22] for mappings. With annotations referring to MLNs, a probabilistic mapping has the form $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma, M)$, where M is the MLN encoding the probabilistic worlds in which the dependencies can either hold or not hold.

As described in [22], the TGDs, negative constraints, and non-conflicting keys are annotated with probabilistic scenarios λ that correspond to the worlds that they are valid in. The complex probabilistic dependencies that the annotations are involved in are represented by the MLN. In the probabilistic extension of Datalog+/– in [22], annotations cannot refer to elements of the databases or the mappings; hence, again, there is a modeling advantage in separating the two tasks of database and mapping dependencies modeling when modeling the uncertainty around the databases and dependencies.

Note that due to the disconnected representation between the probabilistic dependencies and the ontology, we can encode a part of mapping formulas as predicates encoding a specific semantics like disjointness, renaming, or decomposition, in a similar way as in [19]. With these predicates, an MLN can be created, and the actual mappings can be enriched by ground predicates that add the probabilistic interpretation. However, another more interesting encoding involves using a second ontology describing additional features of the generation of the mappings, and in this way eventually even meta reasoning about the mapping generation is possible. A rather general example of such an additional MLN describing the generation of a mapping is shown in Fig. 1. Here, the MLN describes the generation of a mapping via the matcher that it generates and a set of (possibly dependent) applicability conditions, as well as additional conditions that influence the probability of the mapping besides the result of the matcher.

With such kind of an MLN describing the dependency of different kinds of conditions (also dependencies between matchers are conceivable to combine the results of several different matchers), probabilistic reasoning over data integration settings can be done in much more precise settings. Hence, such an annotation is clearly much more expressive than the one corresponding to the two cases considered in Section 7.

7 Computational Complexity

In this section, we explore the computational complexity of deciding the existence of (universal) solutions for deterministic and probabilistic data exchange problems in our framework with mappings encoded in different variants of Datalog+/–.

We assume that all annotations are in disjunctive normal form (DNF), i.e., disjunctions of conjunctions of literals, and we consider the following two cases: (i) that elementary events and their negations are

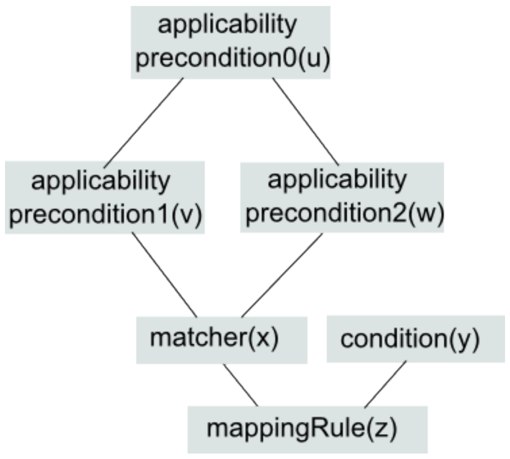
applicability precondition0(u)
applicability precondition1(v)
applicability precondition2(w)
matcher(x)
condition(y)
mappingRule(z)

Figure 1: Example of a Markov logic network describing the generation of mappings by means of applicability conditions and an additional condition that influences the probability of the mapping besides the result of the matcher.

pairwise probabilistically independent (i.e., the probability of worlds $\ell_1 \wedge \cdots \wedge \ell_n$ of elementary events ($\ell_i = e_i$) and their negations ($\ell_i = \neg e_i$) is defined as $\Pi_{i=1}^{n} \nu(\ell_i)$, where $\nu(\ell_i) = \mu(e_i)$ and $\nu(\ell_i) = 1 - \mu(e_i)$, respectively), called *elementary-event-independence*; and (ii) that all annotations are also elementary and that all worlds have a positive probability, called *tuple-independence*.

7.1 Deterministic Data Exchange

In the following, we show that deciding the existence of probabilistic (or probabilistic universal) solutions for deterministic data exchange problems relative to probabilistic source databases is co-NP-complete (resp., in PTIME) in the elementary-event-independent (resp., tuple-independent) case in the data complexity.

Theorem 1. *Given a schema mapping $\mathcal{M} = (\boldsymbol{S}, \boldsymbol{T}, \Sigma_{st} \cup \Sigma_t)$, where Σ_{st} and Σ_t are guarded TGDs, negative constraints, and non-conflicting keys over $\boldsymbol{S} \cup \boldsymbol{T}$ and $\boldsymbol{T}$, respectively, and a probabilistic source database Pr_s, deciding whether there exists a probabilistic (or probabilistic universal) solution for Pr_s relative to $\mathcal{M}$ is* co-NP*-complete in the elementary-event-independent case in the data complexity.*

Proof. From Proposition 4.3 in [9] it follows that a probabilistic solution $Pr_t = (\mathcal{J}, \mu_t)$ for a probabilistic source database $Pr_s = (\mathcal{I}, \mu_s)$ relative to a schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ exists iff a deterministic solution exists for all worlds of the probabilistic source database relative to the schema mapping. In addition, it also follows that a probabilistic universal solution exists iff a universal solution exists.

To prove the upper bound for the elementary-event-independent case, we decide the complementary problem by guessing a world and checking that it is not a deterministic solution relative to its database, which is in NP.

The lower bound for the elementary-event-independent case follows from a polynomial reduction from the co-NP-complete problem of deciding whether a CNF formula $\phi = c_1 \wedge \cdots \wedge c_n$ is unsatisfiable. In a CNF formula $\phi = c_1 \wedge \cdots \wedge c_n$, every c_i is a disjunction of literals over m propositional variables $x_1, \ldots, x_m$. In the following, we construct a fixed schema mapping that is independent of ϕ while ϕ defines the source instance with the propositional variables being the elementary events.

The source database in this reduction has a binary relation symbol E_S and a unary relation symbol P_S. The source database consists of the atoms $E_S(i-1, i)$ for all $i \in \{1, \dots, n\}$, annotated with c_i, and the atom $P_S(0)$, annotated with the true event $\top$, while the probabilities of the variables x_i are defined as 0.5.

Similarly to the source schema **S**, the target schema **T** consists of a binary relation symbol E_T and a unary relation symbol P_T. The set Σ in the deterministic mapping in Datalog+/– is defined as $m_1 : E_S(X, Y) \rightarrow E_T(X, Y)$, $m_2 : P_S(X) \rightarrow P_T(X)$, $m_3 : P_T(n) \rightarrow \bot$, and $m_4 : P_T(X) \wedge E_T(X, Y) \rightarrow P_T(Y)$.

Next, we will show that the above probabilistic database and deterministic mapping have a probabilistic solution iff ϕ is unsatisfiable.

Given a truth assignment τ for ϕ, we denote by G_τ the directed graph with the node set {0, 1, ..., n} and the edge $(i-1, i)$ for each conjunct c_i of ϕ that is satisfied by the truth assignment τ. Note that the E_S relation of the source instance contains exactly the edges of G_τ. Observe also that G_τ contains a path from 0 to n iff all the conjuncts of ϕ are satisfied. So, it is enough to prove that there exists a solution for the probabilistic source database iff G_τ does not contain a path from 0 to m, for all τ.

A solution for the probabilistic database (when it exists) is a directed graph G', where the edges are given by E_T. P_T can be considered a labeling of nodes. m_1 implies that the solution G' must contain all the edges of G_τ; m_2 and m_4 imply that starting from a labeling of node 0 which is transferred to G' via m_2, all nodes involved in a path from node 0 are labelled via m_4. m_3, however, prohibits node n to be labeled.

Obviously, the labeling with P_T is propagated through paths in G_τ by means of the mapping. Hence, if G_τ contains a path from 0 to n, we get a contradiction to the fact that they are not both labeled in G'. On the other hand, if G_τ does not contain a path from 0 to n, then we can obtain G' from G_τ by labeling with P_T all the nodes that are reachable from 0 with E_T. It follows that G' exists iff no path of G_τ leads from 0 to m, for all τ. □

Observe that by the construction presented in the proof of Theorem 1, hardness for co-NP also holds for the special case where the constructed mapping is formulated without existentially quantified variables.

Next we prove that deciding whether there exists a probabilistic (or probabilistic universal) solution for a tuple-independent probabilistic source database relative to a mapping is tractable.

Theorem 2. *Given a schema mapping $\mathcal{M} = ($**S**, **T**, $\Sigma_{st} \cup \Sigma_t)$, where Σ_{st} and Σ_t are guarded TGDs, negative constraints, and non-conflicting keys over* **S** $\cup$ **T** *and* **T***, respectively, and a probabilistic source database Pr_s, deciding whether there exists a probabilistic (or probabilistic universal) solution for Pr_s relative to $\mathcal{M}$ is in* PTIME *in the tuple-independent case in the data complexity.*

Proof. From Proposition 4.3 in [9] it follows that a probabilistic solution $Pr_t = (\mathcal{J}, \mu_t))$ for a probabilistic source database $Pr_s = (\mathcal{I}, \mu_s)$ relative to a schema mapping $\mathcal{M} = ($**S**, **T**, $\Sigma)$ exists iff a deterministic solution exists for all worlds of the probabilistic source database relative to the schema mapping. In addition, it also follows that a probabilistic universal solution exists iff a universal solution exists.

In the tuple-independent case, the tractability follows from the fact that a deterministic solution exists relative to the maximal possible deterministic source database, which can be decided in polynomial time in the size of the database. □

The next result shows that deciding the existence of probabilistic (or probabilistic universal) solutions for deterministic data exchange problems relative to probabilistic source databases is also in PTIME in the elementary-event-independent case in the data complexity, if we restrict the TGDs in the mapping to linear rather than guarded TGDs. The main idea behind this result is that, in the linear case, all the target inconsistencies can be traced back to a polynomial number of potential source inconsistencies, which must be associated with annotations that are either inconsistent or have probability zero, which can be checked in polynomial time in the data complexity.

Theorem 3. *Given a schema mapping $\mathcal{M} = (\boldsymbol{S}, \boldsymbol{T}, \Sigma_{st} \cup \Sigma_t)$, where Σ_{st} and Σ_t are linear TGDs, negative constraints, and non-conflicting keys over $\boldsymbol{S} \cup \boldsymbol{T}$ and $\boldsymbol{T}$, respectively, and a probabilistic source database Pr_s, deciding whether there exists a probabilistic (or probabilistic universal) solution for Pr_s relative to $\mathcal{M}$ is in* PTIME *in the elementary-event-independent case in the data complexity.*

Proof. We base the proof on the observation that without inconsistencies, we yield a solution with the chase and, consequently, also a probabilistic (universal) solution. Observe also that the equality generating dependencies are not involved in any inconsistencies because we require them to be non-conflicting, cf. section 2, which means that their validity can be checked on the database. Hence, inconsistencies can only arise through negative constraints in Σ_{st} and in Σ_t.

The maximum numbers of inconsistencies occur when all negative constraints are involved. In the inconsistency caught by a negative constraint all atoms in its body are involved. These atoms can be traced backwards via a chain of target TGDs and finally source-to-target TGDs to a polynomial number of potential source inconsistencies in the source database. Due to the linearity of the tuple generating dependencies, an atom in a negative constraint is only swapped by another atom which is the body atom $b\theta$ of a tgd whose head has a most general unifyer θ with the original atom in the negative constraint. Hence, an exponential blow up during the backtrace is impossible.

Let l be the number of negative constraints in $\Sigma_{st} \cup \Sigma_t$. Let k be the maximum number of atoms occuring in a negative constraint in the set $\Sigma_{st} \cup \Sigma_t$. Let m be the maximum number of arguments in a predicate occuring in the l negative constraints. Let p be the number of predicates occuring in the database and in the sets $\Sigma_{st} \cup \Sigma_t$. Let c be the number of constants occuring in the database and $c_{nc,TGDs}$ the constants occuring in the negative constraints and the TGDs in $\Sigma_{st} \cup \Sigma_t$. Then, the upper bound on the number of possible atoms generated by tracing back from negative constraints to the database through TGDs is $d = \sum_{l=1}^{k} p^l (c_{nc,TGDs} + lm)^{lm}$.

The constant d is providing an upper bound for the generation of atoms through the backtrace. During the final unification with database atoms, in the worst case polynomial many database atoms can be unified with the atoms in the negative constraints. The polynomial number of atoms is then bounded by the constant d in the following way: $c^m d$ with $|c| \leq ||db||$.

The annotations λ_i associated with the atoms a_i being traced back must either have probability zero or be inconsistent in order to yield a (probabilistic (universal)) solution. We first check whether the probability of the annotation of an inconsistency is zero. If it is not zero, we then check whether it is inconsistent.

In order to check the annotation of an inconsistency, we first have to construct it via collecting all source database atoms involved in a possible inconsistency with the negative constraint nc_i with $i \in \{1, \dots, l\}$. After having collected the atoms, we join their annotations λ_j with $j \in \{1, \dots, k\}$ via conjunction. Note that the annotations of the source database atoms are in DNF format and, hence, we have a conjunction of formulas in DNF. When we transform this conjunction of formulas in DNF to the disjunctive normal form (DNF), we yield a disjunction of n^k conjuncts with n being the maximum number of conjuncts in the annotations of the k atoms of the current negative constraint. Hence, this transformation is done in polynomial time in the length of the annotations. We need to do such a transformation for each negative constraint yielding ϕ_i formulas with $i \in \{1, \dots, l\}$ and obviously remain in PTIME.

After we made sure that each ϕ_i with $i \in \{1, \dots, l\}$ is in DNF, we first check for each ϕ_i whether its probability is 0; i.e. whether the probability of each of the conjuncts is 0. This can be done in linear time in the length of the formula as for each of the conjuncts, we just need to multiply the probability of each involved event literal and at least one of them needs to be zero. In case the probability of a ϕ_i is not 0, we then check whether ϕ_i is inconsistent which can be done in linear time (in the length of the formula) as well because it is represented in DNF format and, hence, we just have to check whether each conjunct contains e and $\neg e$ with e being an arbitrary event (possibly a different e in each conjunct). All checks obviously remain in PTIME.

It follows that in case all ϕ_i have either probability 0 or are inconsistent, we have a solution and, consequently, also a probabilistic (universal) solution. Otherwise, we neither have a deterministic solution nor a probabilistic (universal) solution. Deciding whether there is a solution is in PTIME as shown above. □

7.2 Probabilistic Data Exchange

Before we discuss complexity results for probabilistic data exchange, note that in probabilistic data exchange, in addition to the probabilistic source database, we also have a probability distribution μ over the set of subsets of deterministic mappings $\Sigma_{st} \cup \Sigma_t$. This means that Definitions 5 and 6 are carried over in a straightforward way to probabilistic mappings. More specifically, each of the probabilistic TGDs, probabilistic negative constraints and probabilistic non-conflicting keys is annotated with an annotation λ as defined in Definition 5. The compact encoding of a probabilistic database carries over directly to probabilistic mappings as well in the spirit of Definition 6 with the probability of each annotation λ being the sum of the probabilities of all worlds in which λ is true and the probability of every *mapping world* $\{\delta_1, \ldots, \delta_n\}$ such that $\{\delta_1 : \lambda_{\delta_1}, \ldots, \delta_n : \lambda_{\delta_n}\}$ with $\delta_i \in \Sigma_{st}$. This means that the compact encoding of a probabilistic data exchange problem consists of both a compact encoding of a probabilistic database and a compact encoding of the probabilistic mapping dependencies as well as probabilistic negative constraints and probabilistic EGDs. Hence, we have the same kind of encoding here like in the deterministic data exchange setting only with the difference that the annotations λ are extended with more events which are used solely for encoding mapping worlds.

The next result shows that deciding the existence of probabilistic (or probabilistic universal) solutions for probabilistic data exchange problems with probabilistic mappings is also co-NP-complete (resp., in PTIME) in the elementary-event-independent case in the data complexity.

The upper bound follows from the fact that relative to a probabilistic source database, a probabilistic universal solution exists iff a probabilistic solution exists, which is in turn equivalent to the existence of a deterministic solution relative to the deterministic database and deterministic mapping for every world. The co-NP-hardness in the probabilistic case follows from the co-NP-hardness in the less general deterministic case.

Theorem 4. *Given a probabilistic schema mapping $\mathcal{M} = (\boldsymbol{S}, \boldsymbol{T}, \Sigma_{st} \cup \Sigma_t, \mu)$, where Σ_{st} and Σ_t are guarded TGDs, negative constraints, and non-conflicting keys over $\boldsymbol{S} \cup \boldsymbol{T}$ and $\boldsymbol{T}$, respectively, and $\mu : 2^{\Sigma_{st}} \rightarrow [0,1]$ and a probabilistic source database Pr_s, deciding whether there exists a probabilistic (or probabilistic universal) solution for Pr_s relative to $\mathcal{M}$ is* co-NP-*complete in the elementary-event-independent case in the data complexity.*

Proof. Note again that from Proposition 4.3 in [9] it follows that a probabilistic solution $Pr_t = (\mathcal{J}, \mu_t)$ for a probabilistic source database $Pr_s = (\mathcal{I}, \mu_s)$ relative to a deterministic schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ exists iff a deterministic solution exists for all worlds of the probabilistic source database relative to the schema mapping. In addition, it also follows that a probabilistic universal solution exists iff a universal solution exists.

To prove the upper bound for the elementary-event-independent case, we decide the complementary problem by guessing a world and checking that it is not a deterministic solution relative to its database, which is in NP.

The lower bound for the elementary-event-independent case follows from a reduction of the deterministic data exchange problem. The probability distribution μ over the set of subsets of deterministic mappings Σ_{st} is assumed to be 1 for the world containing all mappings and 0 for all subsets thereof. Hence, deterministic data exchange corresponds to a special case of probabilistic data exchange which proves hardness in co-NP. □

The next result shows that deciding the existence of probabilistic (or probabilistic universal) solutions for probabilistic data exchange problems with probabilistic source databases is in PTIME in the tuple-independent case in the data complexity. The tractability follows from the fact that a probabilistic solution exists relative to a probabilistic source database iff a deterministic solution exists relative to the maximal possible deterministic source database and mapping, which can be decided in polynomial time.

Theorem 5. *Given a probabilistic schema mapping $\mathcal{M} = (\boldsymbol{S}, \boldsymbol{T}, \Sigma_{st} \cup \Sigma_t, \mu)$, where Σ_{st} and Σ_t are guarded TGDs, negative constraints, and non-conflicting keys over $\boldsymbol{S} \cup \boldsymbol{T}$ and $\boldsymbol{T}$, respectively, and $\mu : 2^{\Sigma_{st}} \to [0,1]$ and a probabilistic source database Pr_s, deciding whether there exists a probabilistic (or probabilistic universal) solution for Pr_s relative to $\mathcal{M}$ is in* PTIME *in the tuple-independent case in the data complexity.*

Proof. Remember that from Proposition 4.3 in [9] it follows that a probabilistic solution $Pr_t = (\mathcal{J}, \mu_t)$ for a probabilistic source database $Pr_s = (\mathcal{I}, \mu_s)$ relative to a deterministic schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ exists iff a deterministic solution exists for all worlds of the probabilistic source database relative to the schema mapping. In addition, it also follows that a probabilistic universal solution exists iff a universal solution exists.

As we are only concerned with data complexity, we assume that $(\Sigma_{st} \cup \Sigma_t, \mu)$ is fixed, yielding $2^{\Sigma_{st}}$ many probabilistic worlds to be considered which is a constant when the mapping is fixed. Hence, tractability follows from the fact that the probabilistic source database is tuple-independent and that a deterministic solution exists relative to the maximal possible deterministic source database which can be decided in polynomial time in the size of the database. □

Like in the deterministic case, deciding the existence of probabilistic (or probabilistic universal) solutions for probabilistic data exchange problems relative to probabilistic source databases is also in PTIME in the elementary-event-independent case in the data complexity, if we restrict the TGDs in the mapping to linear rather than guarded TGDs. This is because we essentially have to consider one deterministic data exchange problem for every subset of the mapping, which is fixed in the data complexity case.

Theorem 6. *Given a probabilistic schema mapping $\mathcal{M} = (\boldsymbol{S}, \boldsymbol{T}, \Sigma_{st} \cup \Sigma_t, \mu)$, where Σ_{st} and Σ_t are probabilistic linear TGDs, probabilistic negative constraints, and probabilistic non-conflicting keys over $\boldsymbol{S} \cup \boldsymbol{T}$ and $\boldsymbol{T}$, respectively, and $\mu : 2^{\Sigma_{st}} \to [0,1]$ and a probabilistic source database Pr_s, deciding whether there exists a probabilistic (or probabilistic universal) solution for Pr_s relative to $\mathcal{M}$ is in* PTIME *in the elementary-event-independent case in the data complexity.*

Proof. Remember that from Proposition 4.3 in [9] it follows that a probabilistic solution $Pr_t = (\mathcal{J}, \mu_t)$ for a probabilistic source database $Pr_s = (\mathcal{I}, \mu_s)$ relative to a schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ exists iff a deterministic solution exists for all worlds of the probabilistic source database relative to the schema mapping. In addition, it also follows that a probabilistic universal solution exists iff a universal solution exists.

As we are only concerned with data complexity, we assume that $(\Sigma_{st} \cup \Sigma_t, \mu)$ is fixed, yielding $2^{\Sigma_{st}}$ many probabilistic worlds to be considered which is a constant when the mapping is fixed. Hence, tractability follows from the proof of Theorem 3 where we showed that in the case of deterministic mappings, the number of source database atoms responsible for a possible inconsistency is bounded by a constant and that their annotations need to be either inconsistent or have probability zero in order to yield a solution. As shown in the proof of Theorem 3, we are able to do all this in polynomial time for the case of deterministic mappings. When we have to deal with probabilistic mappings, we have to repeat the procedure proposed in the proof of Theorem 3 for each of the $2^{\Sigma_{st}}$ possible worlds of the mappings of which there is a constant number when we are considering solely data complexity. □

8 Provenance

With the compact encoding of probabilistic databases via annotated atoms with events representing the worlds in which they are valid, we are essentially using a data provenance formalism for tracing the probabilistic worlds that an atom belongs to.

Provenance information adds value to data by explaining how it was obtained, thus allowing to validate its correctness, truthfulness, trustworthiness, and compliance. In information integration, when pieces of data from distributed databases or ontologies are integrated, provenance information allows to check the trustworthiness and correctness of the results of queries and debug them as well as trace the errors back to where they were created. Hence, an information integration framework should be equipped with some form of provenance information.

Provenance research distinguishes between *workflow* and *data* provenance. The former is about capturing the processes (i.e., flows and transformations) that a piece of data has gone through before arriving at its current destination or its current version [23]. Some of these processes cannot be accessed and remain a black box in workflow provenance. Contrary to workflow provenance, data provenance is much more fine-grained and focuses on the lineage of data for a query – i.e., the whereabouts of its derivation in a database itself. While we are mainly interested in data provenance, workflow provenance certainly is also of much relevance for data exchange. Hence, the recent W3C recommendation PROV [24], which is a language for specifying workflow provenance in the Web, is relevant to our work as well. However, as we have a fine-grained logical formalization of the data exchange process, we focus on data provenance.

In data provenance, there is a principal distinction made among where-, why- and how-provenance [25]. How-provenance [26] is the most expressive one and the most appropriate for annotating mappings and tracing back the origin of query results. How-provenance can be modeled via an algebraic structure, called a *semiring*, and it is possible to construct different kinds of semirings, depending on what kind of information is to be captured, and which operations on that information are to be allowed. Besides formalizing different kinds of provenance annotations with certain kinds of semirings (called K-relations) based on the positive relational algebra, [26] provides a formalization of plain Datalog without negation with K-relations, used within the collaborative data sharing system ORCHESTRA [27] also for modeling TGDs without existential quantifiers. To capture applications of mappings in ORCHESTRA, [28] proposes to use a so-called $\mathcal{M}$-semiring, which allows to annotate the mappings with names $m_1, \ldots, m_k$ (unary functions), one for each mapping. This can be combined with the formalization of negation-free Datalog (with a procedural semantics based on the least fixpoint operator to construct the model) with positive K-relations as presented in [26].

Clearly, such a formalization for the probabilistic Datalog+/– information integration framework in this paper allows to capture provenance and annotate the mappings with an id such that the integration paths can be traced back to their origin. In this way, routes that can be used to debug mappings like in [29] can be captured. In addition, as shown in [26], when the mappings are the only probabilistic or uncertain elements, the probabilities can also be computed more efficiently, as the captured provenance also carries the information where the probabilities are propagated from. In addition, cycles can be detected, and the trustworthiness of query results can also be estimated, as it can be detected where the data that is involved in the query result has been integrated from. For this purpose, the trustworthiness of data sets and possibly also peers who provide access to data sets need to be assessed beforehand.

A similar approach to the aforementioned ORCHESTRA system for the application of the chase within probabilistic Datalog+/– with a semiring formalization can be constructed and is currently under development. In probabilistic data integration with Datalog+/–, lineage is restricted by the guards, which help to direct the chase towards the answer of a query through the annotated guarded chase forest. In [30, 22], a similar kind of tuple annotation as proposed here was used in combination with the chase to speed up the reasoning process.

9 Related Work

Probabilistic data exchange with different combinations of source-to-target TGDs with and without existential quantified variables in the head, target equality constraints, and weakly-acyclic target TGDs with and without existential quantified variables in the head have been studied in [9]. In contrast, Datalog+/– allows to deal with ontologies on the Semantic Web and as such allows to integrate information residing in ontologies. One work on integrating information in ontologies is [31], which tackles the problem of knowledge base data exchange; however, they assume that ontologies in *DL-Lite* are used to exchange data – guarded Datalog+/– strictly subsumes *DL-Lite* and goes well beyond its expressive power.

Other articles that are closely related to our work and that of [9] are [32, 33]. There, the source data is deterministic and the mappings are probabilistic. By-table and by-tuple solutions are defined; while the former correspond to a restriction of general probabilistic mappings, by-tuple solutions correspond to mappings that translate only single facts and hence correspond to inclusion dependencies.

In the Semantic Web, the integration of information in ontologies has also been addressed in [19]. There, predicates are defined encoding specific semantics like disjointness, renaming, or decomposition, and an MLN is built from them. Very much related is also our prior work in [34] and [35]. There, ontologies are mapped with Bayesian description logic programs, which correspond to a subset of probabilistic Datalog+/– and tightly coupled description logics programs with negation under different semantics like the answer set semantics and the well-founded semantics.

Provenance for information integration is used in ORCHESTRA [27] for integrating XML data. In contrast, we exchange data between databases, and we also deal with uncertain and incomplete databases.

10 Summary and Outlook

In this paper, we have studied probabilistic data exchange via probabilistic Datalog+/– and used annotations encoding probabilistic data provenance of atoms. We have also considered using provenance for tracking the origin of integrated information and using provenance for tracking the origin of mappings themselves, including the conditions for their applicability for trust assessment and debugging.

By means of Datalog+/– [10], which can represent *DL-Lite* and $\mathcal{EL}$, we use a tractable language with dependencies that allows to nicely tie together the theoretical results on information integration in databases and the work on ontology mediation in the Semantic Web. The separation between the ontology and the probabilistic dependencies allows us to either model the mappings with specific newly-invented predicates like disjointness, renaming, or decomposition, etc. or – more interestingly – with a probabilistic meta ontology describing the matching process.

Our work shows how classical and probabilistic (guarded and linear) Datalog+/– can be used to model information integration settings, and sketches a deterministic mapping language based on Datalog+/– and a probabilistic generalization based on the rather loosely coupled probabilistic extension of Datalog+/– with worlds represented by propositional events. We also justify why data provenance needs to be captured and represented within such a probabilistic information integration framework, and propose to use an adaptation of K-relations as proposed by [26]. Such an extension with provenance allows to track how results of queries to the framework have been created, and also debug mappings, since errors can be traced back to their origin.

An interesting topic for future research is to develop the proposed framework for provenance capture and, among others, investigate how to use the chase for reasoning with probabilistic Datalog+/– within a semiring-framework.

References

[1] T. Lukasiewicz, L. Predoiu, Information integration with provenance on the Semantic Web via probabilistic Datalog+/–, in: Proceedings of URSW, CEUR Workshop Proceedings, 2013.

[2] T. Lukasiewicz, M. V. Martinez, L. Predoiu, G. I. Simari, Information Integration with Provenance on the Semantic Web via Probabilistic Datalog+/-, Lecture Notes in Computer Science, Springer, 2014.

[3] L. M. Haas, Beauty and the beast: The theory and practice of information integration, in: Proceedings of ICDT, Vol. 4353 of LNCS, Springer, 2007, pp. 28–43.

[4] P. A. Bernstein, L. M. Haas, Information integration in the enterprise, Commun. ACM 9 (51) (2008) 72–79.

[5] M. Lenzerini, Data integration: A theoretical perspective, in: Proceedings of PODS, ACM Press, 2002, pp. 233–246.

[6] R. Fagin, P. G. Kolaitis, R. J. Miller, L. Popa, Data exchange: Semantics and query answering, Theor. Comput. Sci. 336 (1) (2005) 89–124.

[7] L. Serafini, H. Stuckenschmidt, H. Wache, A formal investigation of mapping language for terminological knowledge, in: Proceedings of IJCAI, Professional Book Center, 2005, pp. 576–581.

[8] R. Fagin, B. Kimelfeld, P. G. Kolaitis, Probabilistic data exchange, in: Proceedings of ICDT, ACM Press, 2010, pp. 76–88.

[9] R. Fagin, B. Kimelfeld, P. G. Kolaitis, Probabilistic data exchange, J. ACM 58.

[10] A. Calì, G. Gottlob, T. Lukasiewicz, A general Datalog-based framework for tractable query answering over ontologies, J. Web Sem. 14 (2012) 57–83.

[11] D. Maier, A. O. Mendelzon, Y. Sagiv, Testing implications of data dependencies, ACM T. Database Syst. 4 (4) (1979) 455–469.

[12] D. S. Johnson, A. Klug, Testing containment of conjunctive queries under functional and inclusion dependencies, J. Comput. System Sci. 28 (1984) 167–189.

[13] A. Calì, G. Gottlob, M. Kifer, Taming the infinite chase: Query answering under expressive integrity constraints, in: Proceedings of KR, AAAI Press, 2008, pp. 70–80.

[14] A. Deutsch, A. Nash, J. Remmel, The chase revisited, in: Proceedings of PODS, ACM Press, 2008, pp. 149–158.

[15] H. Andréka, I. Németi, J. van Benthem, Modal languages and bounded fragments of predicate logic, J. Philos. Logic 27 (3) (1998) 217–274.

[16] H. J. Levesque, R. J. Brachman, Expressiveness and tractability in knowledge representation and reasoning, Comput. Intell. 3 (1987) 78–93.

[17] B. ten Cate, P. G. Kolaitis, Structural characterizations of schema-mapping languages, Commun. ACM 53 (2010) 101–110.

[18] J. Euzenat, P. Shvaiko, Ontology Matching, Springer, 2007.

[19] M. Niepert, J. Noessner, C. Meilicke, H. Stuckenschmidt, Probabilistic logical Web data integration, in: Reasoning Web, Vol. 6848 of LNCS, Springer, 2011, pp. 504–533.

[20] F. Scharffe, J. de Bruijn, A language to specify mappings between ontologies, in: Proceedings of SITIS, Dicolor Press, 2005, pp. 267–271.

[21] R. Fagin, P. G. Kolaitis, L. Popa, W.-C. Tan, Composing schema mappings: Second-order dependencies to the rescue, ACM T. Database Syst. 30 (2005) 994–1055.

[22] G. Gottlob, T. Lukasiewicz, M. V. Martinez, G. I. Simari, Query answering under probabilistic uncertainty in Datalog+/– ontologies, Ann. Math. Artif. Intell. 69 (1) (2013) 37–72. doi:10.1007/s10472-013-9342-1.

[23] P. Buneman, The providence of provenance, in: Proceedings of BNCOD, Vol. 7968 of LNCS, Springer, 2013, pp. 7–12.

[24] L. Moreau, P. Groth, Provenance: An Introduction to PROV, Morgan and Claypool, 2013.

[25] J. Cheney, L. Chiticariu, W.-C. Tan, Provenance in databases: Why, how and where, Foundation and Trends in Databases 1 (4) (2009) 379–474.

[26] T. J. Green, G. Karvounarakis, V. Tannen, Provenance semirings, in: Proceedings of PODS, ACM Press, 2007, pp. 31–40.

[27] T. J. Green, G. Karvounarakis, N. E. Taylor, O. Biton, Z. G. Ives, V. Tannen, ORCHESTRA: Facilitating collaborative data sharing, in: Proceedings of SIGMOD, ACM Press, 2007, pp. 1131–1133.

[28] G. Karvounarakis, Provenance in collaborative data sharing, Ph.D. thesis, University of Pennsylvania (2009).

[29] L. Chiticariu, W. C. Tan, Debugging schema mappings with routes, in: Proceedings of VLDB, ACM Press, 2006, pp. 79–90.

[30] T. Lukasiewicz, M. V. Martinez, G. I. Simari, Consistent answers in probabilistic Datalog+/– ontologies, in: Proceedings of RR, Vol. 7497 of LNCS, Springer, 2012, pp. 156–171.

[31] E. Botoeva, Description logic knowledge base exchange, Ph.D. thesis, Free University of Bozen-Bolzano (2014).

[32] X. L. Dong, A. Halevy, C. Yu, Data integration with uncertainty, VLDB J. 18 (2009) 469–500.

[33] A. Gal, M. V. Martinez, G. I. Simari, V. S. Subrahmanian, Aggregate query answering under uncertain schema mappings, in: Proc. of ICDE, IEEE Computer Society, 2009, pp. 940–951.

[34] T. Lukasiewicz, L. Predoiu, H. Stuckenschmidt, Tightly integrated probabilistic description logic programs for representing ontology mappings, Ann. Math. Artif. Intell. 63 (3/4) (2011) 385–425.

[35] A. Calì, T. Lukasiewicz, L. Predoiu, H. Stuckenschmidt:, Rule-based approaches for representing probabilistic ontology mappings, Vol. 5327 of LNCS, Springer, 2008, pp. 66–87.