Information Integration with Provenance on the Semantic Web via Probabilistic Datalog$^{\pm}$

Thomas Lukasiewicz and Livia Predoiu

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

Abstract. The recently introduced Datalog$^{\pm}$ family of tractable knowledge representation formalisms is able to represent and reason over light-weight ontologies. It extends plain Datalog by negative constraints and the possibility of rules with existential quantification and equality in rule heads, and at the same time restricts the rule syntax by the addition of so-called guards in rule bodies to gain decidability and tractability. In this paper, we investigate how a recently proposed probabilistic extension of Datalog$^{\pm}$ can be used for representing ontology mappings in typical information integration settings, such as data exchange, data integration, and peer-to-peer integration. To allow to reconstruct the history of the mappings, to detect cycles, and to enable mapping debugging, we also propose to extend it by provenance annotations.

1 Introduction

Information integration aims at querying in a uniform way information that is distributed over multiple heterogeneous sources. This is usually done via mappings between logical formalizations of data sources such as database schemas, ontology schemas, or TBoxes; see also [16, 9, 21]. It is commonly agreed that there are mainly three principles on how data or information from different sources can be integrated:

- **Data exchange:** Data structured under a source schema S (or more generally under 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.

Recently, a probabilistic extension of Datalog$^{\pm}$ [11] has been introduced, which we here propose to use as a mapping language in the above information integration scenarios. Classical Datalog$^{\pm}$ [2] combines Datalog with negative constraints and tuple- and

equality-generating dependencies (TGDs and EGDs, respectively) under certain restrictions to gain decidability and data tractability. In this way, it is possible to capture the *DL-Lite* family of description logics and also the description logic $\mathcal{EL}$. The probabilistic extension is based on Markov logic networks (MLNs) [19].

In this paper, we investigate how probabilistic Datalog$^{\pm}$ can be used as a mapping language for information integration and propose to add provenance information to mappings 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. Furthermore, it helps to detect whether and how to perform mapping updates in case 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.

2 Guarded Datalog$^{\pm}$

We now describe guarded Datalog$^{\pm}$ [2], 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$^{\pm}$ ontologies.

2.1 Databases and Queries

For the elementary ingredients, we assume data constants, nulls, and variables as follows; they serve as arguments in atomic formulas in databases, queries, and dependencies. We assume (i) an infinite universe of *data constants* Δ (which constitute the "normal" domain of a database), (ii) an 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) an infinite set of variables $\mathcal{V}$ (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 $P(t_1, ..., t_n)$, where P is an n-ary predicate, and $t_1, ..., t_n$ are terms. We denote by $pred(\mathbf{a})$ and $dom(\mathbf{a})$ 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 $Q(\mathbf{X}) = \exists \mathbf{Y}\, \Phi(\mathbf{X}, \mathbf{Y})$, where $\Phi(\mathbf{X}, \mathbf{Y})$ is a conjunction of atoms with the variables $\mathbf{X}$ and $\mathbf{Y}$, and eventually constants, 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 $\mu\colon \Delta \cup \Delta_N \cup \mathcal{V} \to \Delta \cup \Delta_N \cup \mathcal{V}$ 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 $Q(D) \neq \emptyset$.

2.2 Tuple-Generating Dependencies

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 $\forall \mathbf{X} \forall \mathbf{Y}\, \Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z})$, 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 $body(\sigma)$ and $head(\sigma)$, 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 $mods(D, \Sigma)$, 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 $ans(Q, D, \Sigma)$, is the set of all tuples $\mathbf{a}$ such that $\mathbf{a} \in Q(B)$ for all $B \in mods(D, \Sigma)$. The *answer* for a BCQ Q to D and Σ is *Yes*, denoted $D \cup \Sigma \models Q$, iff $ans(Q, D, \Sigma) \neq \emptyset$. 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.

2.3 The Chase

The *chase* was introduced to enable checking implication of dependencies [17] and later also for checking query containment [14]. 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 $chase^0(D, \Sigma)$, 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 $chase^k(D, \Sigma)$, 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) $I_1, \ldots, I_n \subseteq chase^{k-1}(D, \Sigma)$ and (ii) the highest level of an atom in every I_i is $k-1$; then, perform every corresponding TGD application on $chase^{k-1}(D, \Sigma)$, 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 $chase(D, \Sigma)$, is then defined as the limit of $chase^k(D, \Sigma)$ for $k \rightarrow \infty$.

The (possibly infinite) chase relative to TGDs is a *universal model*, i.e., there exists a homomorphism from $chase(D, \Sigma)$ onto every $B \in mods(D, \Sigma)$ [3, 7]. 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 $chase(D, \Sigma) \models Q$. In the case of guarded TGDs Σ, such BCQs Q can be evaluated on an initial fragment of $chase(D, \Sigma) \models Q$ 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 [1]. Note also that guardedness is a truly fundamental class ensuring decidability as adding a single unguarded Datalog rule to a guarded Datalog$^{\pm}$ program may destroy decidability as shown in [3].

2.4 Negative Constraints

Another crucial ingredient of Datalog$^{\pm}$ 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 positive, otherwise the negative constraints can be simply ignored when answering the original BCQ Q.

2.5 Equality-Generating Dependencies

A further important ingredient of Datalog$^{\pm}$ 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 $body(\sigma)$, 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 $head(\sigma)$. 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 $chase(D, \Sigma_T \cup \Sigma_E)$, 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* [2]. 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).

2.6 Guarded Datalog$^{\pm}$ Ontologies

We define (guarded) Datalog$^{\pm}$ ontologies as follows. A *(guarded) Datalog$^{\pm}$ 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 Probabilistic Datalog$^{\pm}$

We consider a probabilistic extension of Datalog$^{\pm}$ based on Markov logic networks (MLNs) [19] as introduced in [11]. We now briefly recall its syntax and semantics.

3.1 Syntax

We assume an infinite universe of data constants Δ, an infinite set of labeled nulls Δ_N, and an infinite set of variables $\mathcal{V}$, as in Datalog$^{\pm}$. Furthermore, we assume a finite set of random variables X, as in MLNs. Informally, a probabilistic guarded Datalog$^{\pm}$ ontology consists of a finite set of probabilistic atoms, guarded TGDs, negative constraints, and separable EGDs, along with an MLN. We provide the formal details next.

We first define the notion of probabilistic scenario. A *(probabilistic) scenario* λ is a (finite) set of pairs (X_i, x_i), where $X_i \in X$, $x_i \in Dom(X_i)$, and the X_i's are pairwise distinct. If $|\lambda| = |X|$, then λ is a *full* probabilistic scenario. If every random variable X_i has a Boolean domain, then we also abbreviate λ by the set of all X_i such that $(X_i, true) \in \lambda$. Intuitively, a probabilistic scenario is used to describe an event in which the random variables in an MLN are compatible with the settings of the random variables described by λ, i.e., each X_i has the value x_i.

If a is an atom, σ_T is a TGD, σ_C is a negative constraint, σ_E is an EGD, and λ is a probabilistic scenario, then: (i) $a : \lambda$ is a *probabilistic atom*; (ii) $\sigma_T : \lambda$ is a *probabilistic TGD (pTGD)*; (iii) $\sigma_C : \lambda$ is a *probabilistic (negative) constraint*; and (iv) $\sigma_E : \lambda$ is a *probabilistic EGD (pEGD)*. We also refer to probabilistic atoms, TGDs, (negative) constraints, and EGDs as *annotated formulas*. Intuitively, annotated formulas hold whenever the events associated with their probabilistic scenarios occur.

A *probabilistic (guarded) Datalog$^{\pm}$ ontology* is a pair $\Phi = (O, M)$, where O is a finite set of probabilistic atoms, guarded TGDs, constraints, and EGDs, and M is an MLN. In the sequel, we implicitly assume that every such $\Phi = (O, M)$ is *separable*, which means that Σ_E^ν is separable from Σ_T^ν, for every $\nu \in Dom(X)$, where Σ_T^ν (resp., Σ_E^ν) is the set of all TGDs (resp., EGDs) σ such that (i) $\sigma : \lambda \in O$ and (ii) λ is contained in the set of all $(X_i, \nu(X_i))$ with $X_i \in X$. As for queries, we are especially interested in the probabilities of the answers of CQs to probabilistic Datalog$^{\pm}$ ontologies, called *probabilistic conjunctive queries* (PCQs).

3.2 Semantics

The semantics of probabilistic Datalog$^{\pm}$ ontologies is given relative to probability distributions over *interpretations* $\mathcal{I} = (D, \nu)$, where D is a database, and $\nu \in Dom(X)$. We say $\mathcal{I}$ *satisfies* an annotated formula $F : \lambda$, denoted $\mathcal{I} \models F : \lambda$, iff whenever $\nu(X) = x$, for all $(X, x) \in \lambda$, then $D \models F$. A *probabilistic interpretation* is a probability distribution Pr over the set of all possible interpretations such that only a finite number of interpretations are mapped to a non-zero value. The probability of an annotated formula $F : \lambda$, denoted $Pr(F : \lambda)$, is the sum of all $Pr(\mathcal{I})$ such that $\mathcal{I} \models F : \lambda$.

Let Pr be a probabilistic interpretation, and $F : \lambda$ be an annotated formula. We say that Pr *satisfies* (or is a *model* of) $F : \lambda$ iff $Pr(F : \lambda) = 1$. Furthermore, Pr is a model of a probabilistic Datalog$^{\pm}$ ontology $\Phi = (O, M)$ iff: (i) Pr satisfies all annotated formulas in O, and (ii) $1 - Pr(\mathit{false} : \lambda) = Pr_M(\lambda)$ for all full probabilistic scenarios λ, where $Pr_M(\lambda)$ is the probability of $\bigwedge_{(X_i, x_i) \in \lambda}(X_i = x_i)$ in the MLN M (and computed in the same way as $P(X = x)$ in MLNs).

As for the semantics of queries, we begin with defining the semantics of PCQs without free variables. Let Φ be a probabilistic Datalog$^{\pm}$ ontology, and Q be a BCQ. The *probability* of Q in Φ, denoted $Pr^{\Phi}(Q)$, is the infimum of $Pr(Q : \{\})$ subject to all probabilistic interpretations Pr such that $Pr \models \Phi$. Note that, as a consequence, the probability of a BCQ Q is the sum of all probabilities of full scenarios where the resulting universal model satisfies Q. We next consider the general case. As usual, given a set of variables V and a set of constants Δ, a *substitution* of V by Δ is a mapping $\theta : V \to \Delta$; given a formula F and substitution θ, we denote by $F\theta$ the formula obtained from F by replacing all variables v_i with $\theta(v_i)$. We can now define answers

to PCQs. Let Φ be a probabilistic Datalog$^{\pm}$ ontology, and Q be a CQ. An *answer* for Q to Φ is a pair (θ, p), where (i) θ is a substitution for the free variables of Q, and (ii) $p \in [0, 1]$ is the probability of $Q\theta$ in Φ. It is *positive* iff $p > 0$.

4 Ontology Mappings with Datalog$^{\pm}$

As a language integrating the description logics and the logic programming paradigm with TGDs, Datalog$^{\pm}$ allows to nicely tie together the theoretical results on information integration in databases and the work on ontology mediation in the Semantic Web.

When integrating data stored in databases or data warehouses, i.e., data organized by database schemas, usually so-called *source-to-target TGDs (s-t TGDs)*, corresponding to so-called *GLAV (global-local-as-view) dependencies*, are used as mappings.

According to [9], a schema mapping is defined as $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, where **S** and **T** are the source and the target schema, respectively, Σ_{st} is the set of source-to-target TGDs and EGDs, and Σ_t is the set of target TGDs and EGDs, respectively.

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

- A **LAV (local as view)** dependency is a source-to-target TGD with a single atom in the body, i.e., it has 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 [4] as essential can also be represented in Datalog$^{\pm}$:

- **Copy (Nicknaming):** Copy a source relation or role (of arbitrary arity n) into a target relation or role (of the same arity n like the source relation or role) and rename it. Note that this kind of mapping is a LAV and a GAV mapping at the same time. Example[1]:

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

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

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

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

[1] Note that all examples are stemming from a consideration of the OAEI benchmark set, more specifically, the ontologies 101 and 301-303.

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

– **Decomposition:** Decompose a source relation or source role (of arbitrary arity n) into two or more target relations or roles or concepts. Note that this is a LAV dependency. Example:

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

Only one mapping construct mentioned in [4] as essential cannot be represented by Datalog$^{\pm}$, and this is the join. 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 $\forall x, y\ S\text{:book}(y), S\text{:person}(x) \rightarrow T\text{:author}(x, y)$ cannot be represented with Datalog$^{\pm}$.

In ontology mediation, the definition of a *mapping* or *alignment* is based on correspondences between so-called matchable entities of two ontologies. The following definition is based on [8]: Let S and T be two ontologies that are to be mapped onto each other; 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 a semantic relation between the two matchable elements. A *mapping* or *alignment* between S and T then is a set of correspondences $C = \cup_{i,j,k}\{\langle e_i, e_j, r_k\rangle\}$ between S and T. Note that this is a very general definition that basically allows to describe any kind of mapping language.

Semantic Web and ontology mapping languages usually contain a subset of the above mentioned mapping expressions and in addition constraints, mainly class disjointness constraints as additional mapping expressions (see also [21, 18]). However, note that both research communities, the data integration and the ontology mediation community, also proposed mapping languages that are also more expressive than even the above mentioned normal source-to-target TGDs, e.g., second-order mappings as described in the requirements of [20] or second-order TGDs [10]. In [18], a probabilistic mapping language based on MLNs that is built by mappings of a couple of basic description logic axioms onto predicates with the desired semantics has been presented. A closer look reveals that the mapping constructs that are used are renaming, decomposition and class disjointness constraints, and combinations thereof.

With Datalog$^{\pm}$, such disjointness constraints can be modeled with NCs Σ_{NC}:

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

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

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

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

EGDs are also part of some mapping languages, especially in the database area, and can be represented by Datalog$^{\pm}$ as long as they are separable from the TGDs. Such

kinds of dependencies allow to create mappings like the one of the following form specifying that publishers of the same book or journal in both, the source and target schema (or ontology), have to be the same:

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

5 Ontology Mappings with Probabilistic Datalog$^{\pm}$

A *probabilistic (guarded) Datalog$^{\pm}$ mapping* has the form $\mathcal{M} = (\mathbf{S}, \mathbf{T}, p\Sigma_{st}, p\Sigma_t, M)$, where (i) **S** and **T** are the source and the target schemas or ontologies, respectively, (ii) $p\Sigma_{st}$ is a set of probabilistic (guarded) TGDs, EGDs, and NCs encoding the probabilistic source-to-target dependencies, (iii) $p\Sigma_t$ is a set of probabilistic (guarded) TGDs, EGDs, and NCs encoding the probabilistic target dependencies, and (iv) M is the MLN encoding the probabilistic worlds.

Observe here that the TGDs, EGDs, and NCs are annotated with probabilistic scenarios λ that correspond to the worlds that they are valid in. The probabilistic dependencies that the annotations are involved in are represented by the MLN. As annotations cannot refer to elements of the ontologies or the mapping except of the MLN itself, there is a modeling advantage of separating the two tasks of ontology modeling and of modeling the uncertainty around the axioms of the ontology.

Note that due to the disconnected representation between the probabilistic dependencies and the ontology, we can encode part of mapping formulas as predicates encoding a specific semantics like disjointness, renaming, or decomposition, in a similar way as done in [18]. 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 consists of using a second ontology describing additional features of the generation of the mappings and in this way eventually even do meta reasoning about the mapping generation. A rather general example of such an additional MLN describing the generation of a mapping is shown in Fig. 1. In this example, the MLN describes the generation of a mapping by means of 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 in order to combine the results of several different matchers), probabilistic reasoning over data integration settings can be done in more precise settings. To our knowledge, such kinds of probabilistic meta ontologies for the matching process have not yet been proposed.

6 Provenance

Data provenance information describes the history of data in its life cycle. It adds value to the data by explaining how it was obtained. In information integration, when data from distributed databases or ontologies is integrated, provenance information allows to check the trustworthiness and correctness of the results of queries and debug them

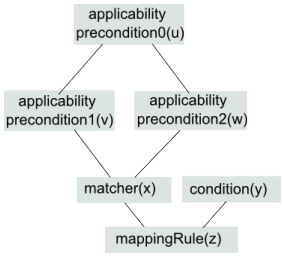

Fig. 1. Example of an MLN describing the generation of mappings by means of applicability conditions and an additional condition that influences the probability of the mapping besides the result of the matcher.

as well as trace the errors back to where they have been created. Hence, an information integration framework should be equipped by some form of provenance.

In data provenance, it is mainly distinguished between where-, why- and how-provenance [5]. How-provenance [12] is the most expressive one and most appropriate for our purpose of annotating mappings and tracing back the origin of query results. How-provenance is modeled by means of a semiring. It is possible to construct different kinds of semirings depending on what kind of information has to be captured and which operations on that information are to be allowed. Besides formalizing different kinds of provenance annotations with a certain kind of semiring (called K-relations) based on the positive relational algebra, [12] provides a formalization of plain Datalog without negation with K-relations that is used within the collaborative data sharing system ORCHESTRA [13] also for modeling TGDs without existential quantifiers. In order to capture applications of mappings in ORCHESTRA, [15] proposes to use a so-called $\mathcal{M}$-semiring, which allows to annotate the mappings with $\mathcal{M} = m_1, \ldots, m_k$ being a set of mapping names, which are 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 [12].

Clearly, such kind of a formalization for our probabilistic Datalog$^{\pm}$ information integration framework would allow 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 [6] can be captured. In addition, as shown in [12], 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.

In order to use a similar approach as the aforementioned ORCHESTRA system, we need to investigate how to model the application of the chase within probabilistic Datalog$^{\pm}$ with a semiring formalization. It can be expected that in probabilistic data

integration with Datalog$^{\pm}$, the lineage will be restricted by the guards who help to direct the chase towards the answer of a query through the annotated guarded chase forest.

7 Summary and Outlook

By means of probabilistic (guarded) Datalog$^{\pm}$ [11], 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.

The paper shows how classical and probabilistic (guarded) Datalog$^{\pm}$ can be used to model information integration settings and sketches a deterministic mapping language based on Datalog$^{\pm}$ and two different kinds of probabilistic adaptations based on the rather loosely coupled probabilistic extension of Datalog$^{\pm}$ with worlds represented by means of an MLN. 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 [12]. Such an extension with provenance allows to track how results of queries to the framework have been created and also debug mappings as errors can be traced back to their origin.

As a next step, we will develop the proposed framework for provenance capture and, amongst others, investigate how to model the chase application for reasoning with probabilistic (guarded) Datalog$^{\pm}$ with a semiring-framework.

Acknowledgments. This work was supported by an EU (FP7/2007-2013) Marie-Curie Intra-European Fellowship, the Engineering and Physical Sciences Research Council (EPSRC) grant EP/J008346/1 "PrOQAW: Probabilistic Ontological Query Answering on the Web", the European Research Council (FP7/2007-2013/ERC) grant 246858 ("DIADEM"), and by a Yahoo! Research Fellowship.

References

1. Andréka, H., Németi, I., van Benthem, J.: Modal languages and bounded fragments of predicate logic. Journal of Philosophical Logic 27(3), 217–274 (1998)
2. Calì, A., Gottlob, G., Lukasiewicz, T.: A general Datalog-based framework for tractable query answering over ontologies. Journal of Web Semantics 14, 57–83 (2012)
3. Calì, A., Gottlob, G., Kifer, M.: Taming the infinite chase: Query answering under expressive integrity constraints. In: Proceedings KR-2008, pp. 70–80. AAAI Press (2008)
4. ten Cate, B., Kolaitis, P.G.: Structural characterizations of schema-mapping languages. Communications of the ACM 53, 101–110 (2010)
5. Cheney, J., Chiticariu, L., Tan, W.C.: Provenance in databases: Why, how and where. Foundation and Trends in Databases 1, 379–474 (2009)
6. Chiticariu, L., Tan, W.C.: Debugging schema mappings with routes. In: Proceedings VLDB-2006, pp. 79–90. ACM Press (2006)

7. Deutsch, A., Nash, A., Remmel, J.: The chase revisited. In: Proceedings PODS-2008, pp. 149–158. ACM Press (2008)
8. Euzenat, J., Shvaiko, P.: Ontology Matching. Springer (2007)
9. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: Semantics and query answering. Theoretical Computer Science 336(1), 89–124 (2005)
10. Fagin, R., Kolaitis, P.G., Popa, L., Tan, W.C.: Composing schema mappings: Second-order dependencies to the rescue. ACM Transactions on Database Systems 30, 994–1055 (2005)
11. Gottlob, G., Lukasiewicz, T., Martinez, M.V., Simari, G.I.: Query answering under probabilistic uncertainty in Datalog+/– ontologies. Annals of Mathematics and Artificial Intelligence (2013)
12. Green, T.J., Karvounarakis, G., Tannen, V.: Provenance semirings. In: Proceedings PODS-2007, pp. 31–40. ACM Press (2007)
13. Green, T.J., Karvounarakis, G., Taylor, N.E., Biton, O., Ives, Z.G., Tannen, V.: ORCHESTRA: Facilitating collaborative data sharing. In: Proceedings SIGMOD-2007, pp. 1131–1133. ACM Press (2007)
14. Johnson, D.S., Klug, A.C.: Testing containment of conjunctive queries under functional and inclusion dependencies. Journal of Computer and System Sciences 28(1), 167–189 (1984)
15. Karvounarakis, G.: Provenance in collaborative data sharing. Ph.D. thesis, University of Pennsylvania (2009)
16. Lenzerini, M.: Data integration: A theoretical perspective. In: Proceedings PODS-2002, pp. 233–246. ACM Press (2002)
17. Maier, D., Mendelzon, A.O., Sagiv, Y.: Testing implications of data dependencies. ACM Transactions on Database Systems (TODS) 4(4), 455–469 (1979)
18. Niepert, M., Noessner, J., Meilicke, C., Stuckenschmidt, H.: Probabilistic Logical Web Data Integration, chap. Reasoning Web. Semantic Technologies for the Web of Data, pp. 504–533. Springer (2011)
19. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning 62(1/2), 107–136 (2006)
20. Scharffe, F., de Bruijn, J.: A language to specify mappings between ontologies. In: Proceedings SITIS-2005, pp. 267–271. IEEE Computer Society (2005)
21. Serafini, L., Stuckenschmidt, H., Wache, H.: A formal investigation of mapping language for terminological knowledge. In: Proceedings IJCAI-2005, pp. 576–581. Professional Book Center (2005)