Tightly Integrated Probabilistic Description Logic Programs for Representing Ontology Mappings

Andrea Calì [1], Thomas Lukasiewicz [2, 3], Livia Predoiu [4], and Heiner Stuckenschmidt [4]

[1] Computing Laboratory, University of Oxford, UK
andrea.cali@comlab.ox.ac.uk
[2] Dipartimento di Informatica e Sistemistica, Sapienza Università di Roma, Italy
lukasiewicz@dis.uniroma1.it
[3] Institut für Informationssysteme, Technische Universität Wien, Austria
lukasiewicz@kr.tuwien.ac.at
[4] Institut für Informatik, Universität Mannheim, Germany
{heiner,livia}@informatik.uni-mannheim.de

Abstract. Creating mappings between ontologies is a common way of approaching the semantic heterogeneity problem on the Semantic Web. To fit into the landscape of semantic web languages, a suitable, logic-based representation formalism for mappings is needed. We argue that such a formalism has to be able to deal with uncertainty and inconsistencies in automatically created mappings. We analyze the requirements for such a formalism, and we propose a novel approach to probabilistic description logic programs as such a formalism, which tightly combines disjunctive logic programs under the answer set semantics with both description logics and Bayesian probabilities. We define the language, and we show that it can be used to resolve inconsistencies and merge mappings from different matchers based on the level of confidence assigned to different rules. Furthermore, we explore the computational aspects of consistency checking and query processing in tightly integrated probabilistic description logic programs. We show that these problems are decidable and computable, respectively, and that they can be reduced to consistency checking and cautious/brave reasoning, respectively, in tightly integrated disjunctive description logic programs. We also analyze the complexity of consistency checking and query processing in the new probabilistic description logic programs in special cases. In particular, we present a special case of these problems with polynomial data complexity.

1 Introduction

The problem of aligning heterogeneous ontologies via semantic mappings has been identified as one of the major challenges of semantic web technologies. In order to address this problem, a number of languages for representing semantic relations between elements in different ontologies as a basis for reasoning and query answering across multiple ontologies have been proposed [27]. In the presence of real world ontologies, it is unrealistic to assume that mappings between ontologies are created manually by domain experts, since existing ontologies, e.g., in the area of medicine contain thousands of concepts and hundreds of relations. Recently, a number of heuristic methods for

matching elements from different ontologies have been proposed that support the creation of mappings between different languages by suggesting candidate mappings (e.g., [10]). These methods rely on linguistic and structural criteria. Evaluation studies have shown that existing methods often trade off precision and recall. The resulting mapping either contains a fair amount of errors or only covers a small part of the ontologies involved [9,11]. To leverage the weaknesses of the individual methods, it is common practice to combine the results of a number of matching components or even the results of different matching systems to achieve a better coverage of the problem [10].

This means that automatically created mappings often contain uncertain hypotheses and errors that need to be dealt with, briefly summarized as follows:

- mapping hypotheses are often oversimplifying, since most matchers only support very simple semantic relations (mostly equivalence between individual elements);
- there may be conflicts between different hypotheses for semantic relations from different matching components and often even from the same matcher;
- semantic relations are only given with a degree of confidence in their correctness.

If we want to use the resulting mappings, we have to find a way to deal with these uncertainties and errors in a suitable way. We argue that the most suitable way of dealing with uncertainties in mappings is to provide means to explicitly represent uncertainties in the target language that encodes the mappings. In this paper, we address the problem of designing a mapping representation language that is capable of representing the kinds of uncertainty mentioned above. We propose an approach to such a language, which is based on an integration of ontologies and rules under probabilistic uncertainty.

There is a large body of work on integrating ontologies and rules, which is a promising way of representing mappings between ontologies. One type of integration is to build rules on top of ontologies, that is, rule-based systems that use vocabulary from ontology knowledge bases. Another form of integration is to build ontologies on top of rules, where ontological definitions are supplemented by rules or imported from rules. Both types of integration have been realized in recent hybrid integrations of rules and ontologies, called *description logic programs* (or *dl-programs*), which have the form $KB = (L, P)$, where L is a description logic knowledge base, and P is a finite set of rules involving either queries to L in a loose integration [6,7] or concepts and roles from L as unary resp. binary predicates in a tight integration [18] (see especially [7,24,18] for detailed overviews on the different types of description logic programs).

Other works explore formalisms for *uncertainty reasoning in the Semantic Web* (an important recent forum for approaches to uncertainty in the Semantic Web is the annual *Workshop on Uncertainty Reasoning for the Semantic Web (URSW)*; there also exists a W3C Incubator Group on *Uncertainty Reasoning for the World Wide Web*). There are especially probabilistic extensions of description logics [15,19], web ontology languages [2,3], and description logic programs [20] (to encode ambiguous information, such as "John is a student with probability 0.7 and a teacher with probability 0.3", which is very different from vague/fuzzy information, such as "John is tall with degree of truth 0.7"). In particular, [20] extends the loosely integrated description logic programs of [6,7] by probabilistic uncertainty as in Poole's independent choice logic (ICL) [26]. The ICL is a powerful representation and reasoning formalism for single- and

also multi-agent systems, which combines logic and probability, and which can represent a number of important uncertainty formalisms, in particular, influence diagrams, Bayesian networks, Markov decision processes, and normal form games. It also allows for natural notions of causes and explanations as in Pearl's structural causal models [13].

In this paper, we propose *tightly integrated probabilistic description logic programs under the answer set semantics* as a language for representing and reasoning with uncertain and possibly inconsistent mappings. The approach is a tight integration of disjunctive logic programs under the answer set semantics, the expressive description logics $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$ (which stand behind the standard web ontology languages OWL Lite and OWL DL [16], respectively), and Bayesian probabilities. More concretely, the tight integration between ontology and rule languages of [18] is combined with probabilistic uncertainty as in the ICL [26]. The resulting language has the following useful features, which will be explained in more detail later:

- The semantics of the language is based on the tight integration between ontology and rule languages of [18], which assumes no structural separation between the vocabularies of the description logic and the logic program components. This enables us to have description logic concepts and roles in both rule bodies and rule heads. This is necessary if we want to use rules to combine ontologies.
- The rule language is quite expressive. In particular, we can have disjunctions in rule heads and nonmonotonic negations in rule bodies. This gives a rich basis for refining and rewriting automatically created mappings for resolving inconsistencies.
- The integration with probability theory provides us with a sound formal framework for representing and reasoning with confidence values. In particular, we can interpret the confidence values as error probabilities and use standard techniques for combining them. We can also resolve inconsistencies by using trust probabilities.
- Consistency checking and query processing in the new rule language are decidable resp. computable, and they can be reduced to their classical counterparts in tightly integrated disjunctive description logic programs. We also analyze the complexity of consistency checking and query processing in special cases, which turn out to be complete for the complexity classes $\mathrm{NEXP}^{\mathrm{NP}}$ and $\mathrm{co\text{-}NEXP}^{\mathrm{NP}}$, respectively.
- There are tractable subsets of the language that are of practical relevance. In particular, we show later that in the case where ontologies are represented in *DL-Lite*, reasoning in the language can be done in polynomial time in the data complexity.

It is important to point out that the probabilistic description logic programs here are very different from the ones in [20] (and their recent tractable variant in [21]). First, they are based on the tight integration between the ontology component L and the rule component P of [18], while the ones in [20,21] realize the loose query-based integration between the ontology component L and the rule component P of [6]. This implies in particular that the vocabularies of L and P here may have common elements (see also Example 4.1), while the vocabularies of L and P in [20,21] are necessarily disjoint. Furthermore, the probabilistic description logic programs here behave semantically very differently from the ones in [20,21] (see Example 4.2). As a consequence, the probabilistic description logic programs here are especially useful for sophisticated probabilistic reasoning tasks involving ontologies (including representing and reasoning with ontology mappings under probabilistic uncertainty and inconsistency), while

the ones in [20,21] can especially be used as query interfaces to web databases (including RDF theories). Second, differently from the programs here, the ones in [20,21] do not allow for disjunctions in rule heads. Third, differently from here, the works [20,21] do not explore the use of probabilistic description logic programs for representing and reasoning with ontology mappings under probabilistic uncertainty and inconsistency.

The rest of this paper is structured as follows. In Section 2, we analyze the requirements of an ontology mapping language. Section 3 briefly reviews description logics as a basis for representing ontologies to be connected by mappings. In Sections 4 and 5, we describe tightly integrated description logic programs as a basis for representing mappings between ontologies as logical rules and explain how the rule language supports the refinement and repair of oversimplifying or inconsistent mappings. Sections 6 and 7 present a probabilistic extension thereof and show that it can be used to represent and combine confidence values of different matchers in terms of error probabilities, and to resolve inconsistencies by using trust probabilities. Sections 8 and 9 address the computational aspects of reasoning in the novel language. In particular, Section 9 identifies a tractable subset of the language. Section 10 concludes with a summary and an outlook. Note that the proofs for the results of this paper are given in Appendix A.

2 Representation Requirements

The problem of ontology matching can be defined as follows [10]. Ontologies are theories encoded in a certain language L. In this work, we assume that ontologies are encoded in OWL DL or OWL Lite. For each ontology O in language L, we denote by $Q(O)$ the matchable elements of the ontology O. Given two ontologies O and O', the task of matching is now to determine correspondences between the matchable elements in the two ontologies. Correspondences are 5-tuples (id, e, e', r, n) such that

- id is a unique identifier for referring to the correspondence;
- $e \in Q(O)$ and $e' \in Q(O')$ are matchable elements from the two ontologies;
- $r \in R$ is a semantic relation (in this work, we consider the case where the semantic relation can be interpreted as an implication);
- n is a degree of confidence in the correctness of the correspondence.

In this paper, we develop a formal language for representing and combining correspondences that are produced by different matching components or systems. From the above general description of automatically generated correspondences between ontologies, we can derive a number of requirements for such a formal language for representing the results of multiple matchers as well as the contained uncertainties:

- *Tight integration of mapping and ontology language:* The semantics of the language used to represent the correspondences between different ontologies has to be tightly integrated with the semantics of the used ontology language (in this case OWL). This is important if we want to use the correspondences to reason across different ontologies in a semantically coherent way. In particular, this means that the interpretation of the mapped elements depends on the definitions in the ontologies.

- *Support for mappings refinement:* The language should be expressive enough to allow the user to refine oversimplifying correspondences suggested by the matching system. This is important to be able to provide a precise account of the true semantic relation between elements in the mapped ontologies. In particular, this requires the ability to describe correspondences that include several elements from the two ontologies.
- *Support for repairing inconsistencies:* Inconsistent mappings are a major problem for the combined use of ontologies because they can cause inconsistencies in the mapped ontologies. These inconsistencies can make logical reasoning impossible, since everything can be derived from an inconsistent ontology. The mapping language should be able to represent and reason about inconsistent mappings in an approximate fashion.
- *Representation and combination of confidence:* The confidence values provided by matching systems is an important indicator for the uncertainty that has to be taken into account. The mapping representation language should be able to use these confidence values when reasoning with mappings. In particular, it should be able to represent the confidence in a mapping rule and to combine confidence values on a sound formal basis.
- *Decidability and efficiency of instance reasoning:* An important use of ontology mappings is the exchange of data across different ontologies. In particular, we normally want to be able to ask queries using the vocabulary of one ontology and receive answers that do not only consist of instances of this ontology but also of ontologies connected through ontology mappings. To support this, query answering in the combined formalism consisting of ontology language and mapping language has to be decidable and there should be efficient algorithms for answering queries at least for relevant cases.

Throughout the paper, we use real data form the Ontology Alignment Evaluation Initiative[1] to illustrate the different aspects of mapping representation. In particular, we use examples from the benchmark and the conference data set. The benchmark dataset consists of five OWL ontologies (tests 101 and 301 to 304) describing scientific publications and related information. The conference dataset consists of about 10 OWL ontologies describing concepts related to conference organization and management. In both cases, we give examples of mappings that have been created by the participants of the 2006 evaluation campaign. In particular, we use mappings created by state-of-the-art ontology matching systems like falcon, hmatch, and coma++.

3 Description Logics

In this section, we recall the description logics $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$, which stand behind the web ontology languages OWL Lite and OWL DL [16], respectively. Intuitively, description logics model a domain of interest in terms of concepts and roles, which represent classes of individuals and binary relations between classes of individuals, respectively. A description logic knowledge base encodes especially subset re-

[1] http://oaei.ontologymatching.org/2006/

lationships between concepts, subset relationships between roles, the membership of individuals to concepts, and the membership of pairs of individuals to roles.

Syntax. We first describe the syntax of $\mathcal{SHOIN}(\mathbf{D})$. We assume a set of *elementary datatypes* and a set of *data values*. A *datatype* is either an elementary datatype or a set of data values (*datatype oneOf*). A *datatype theory* $\mathbf{D} = (\Delta^{\mathbf{D}}, \cdot^{\mathbf{D}})$ consists of a *datatype domain* $\Delta^{\mathbf{D}}$ and a mapping $\cdot^{\mathbf{D}}$ that assigns to each elementary datatype a subset of $\Delta^{\mathbf{D}}$ and to each data value an element of $\Delta^{\mathbf{D}}$. The mapping $\cdot^{\mathbf{D}}$ is extended to all datatypes by $\{v_1, \ldots\}^{\mathbf{D}} = \{v_1^{\mathbf{D}}, \ldots\}$. Let $\mathbf{A}$, $\mathbf{R}_A$, $\mathbf{R}_D$, and $\mathbf{I}$ be pairwise disjoint (denumerable) sets of *atomic concepts*, *abstract roles*, *datatype roles*, and *individuals*, respectively. We denote by $\mathbf{R}_A^-$ the set of *inverses* R^- of all $R \in \mathbf{R}_A$.

A *role* is any element of $\mathbf{R}_A \cup \mathbf{R}_A^- \cup \mathbf{R}_D$. *Concepts* are inductively defined as follows. Every $\phi \in \mathbf{A}$ is a concept, and if $o_1, \ldots, o_n \in \mathbf{I}$, then $\{o_1, \ldots, o_n\}$ is a concept (*oneOf*). If ϕ, ϕ_1, and ϕ_2 are concepts and if $R \in \mathbf{R}_A \cup \mathbf{R}_A^-$, then also $(\phi_1 \sqcap \phi_2)$, $(\phi_1 \sqcup \phi_2)$, and $\neg\phi$ are concepts (*conjunction*, *disjunction*, and *negation*, respectively), as well as $\exists R.\phi$, $\forall R.\phi$, $\geqslant nR$, and $\leqslant nR$ (*exists*, *value*, *atleast*, and *atmost restriction*, respectively) for an integer $n \geqslant 0$. If D is a datatype and $U \in \mathbf{R}_D$, then $\exists U.D$, $\forall U.D$, $\geqslant nU$, and $\leqslant nU$ are concepts (*datatype exists*, *value*, *atleast*, and *atmost restriction*, respectively) for an integer $n \geqslant 0$. We write $\top$ and $\bot$ to abbreviate the concepts $\phi \sqcup \neg\phi$ and $\phi \sqcap \neg\phi$, respectively, and we eliminate parentheses as usual.

An *axiom* has one of the following forms: (1) $\phi \sqsubseteq \psi$ (*concept inclusion axiom*), where ϕ and ψ are concepts; (2) $R \sqsubseteq S$ (*role inclusion axiom*), where either $R, S \in \mathbf{R}_A \cup \mathbf{R}_A^-$ or $R, S \in \mathbf{R}_D$; (3) $\mathrm{Trans}(R)$ (*transitivity axiom*), where $R \in \mathbf{R}_A$; (4) $\phi(a)$ (*concept membership axiom*), where ϕ is a concept and $a \in \mathbf{I}$; (5) $R(a, b)$ (resp., $U(a, v)$) (*role membership axiom*), where $R \in \mathbf{R}_A$ (resp., $U \in \mathbf{R}_D$) and $a, b \in \mathbf{I}$ (resp., $a \in \mathbf{I}$ and v is a data value); and (6) $a = b$ (resp., $a \neq b$) (*equality* (resp., *inequality*) *axiom*), where $a, b \in \mathbf{I}$. A *(description logic) knowledge base* L is a finite set of axioms. For decidability, number restrictions in L are restricted to simple abstract roles [17].

The syntax of $\mathcal{SHIF}(\mathbf{D})$ is as the above syntax of $\mathcal{SHOIN}(\mathbf{D})$, but without the oneOf constructor and with the atleast and atmost constructors limited to 0 and 1.

Example 3.1. A university database may use a knowledge base L to characterize students and exams. For example, suppose that (1) every bachelor student is a student; (2) every master student is a student; (3) every student is either a bachelor student or a master student; (4) professors are not students; (5) only students give exams and only exams are given; (6) *mary* is a student, *john* is a master student, *java* is an exam, and *john* has given it. These relationships are expressed by the following axioms in L:

(1) $bachelor\_student \sqsubseteq student$; (2) $master\_student \sqsubseteq student$;
(3) $student \sqsubseteq bachelor\_student \sqcup master\_student$; (4) $professor \sqsubseteq \neg student$;
(5) $\geqslant 1\ given \sqsubseteq student$; $\geqslant 1\ given^{-1} \sqsubseteq exam$;
(6) $student(mary)$; $master\_student(john)$; $exam(java)$; $given(john, java)$.

Semantics. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ relative to a datatype theory $\mathbf{D} = (\Delta^{\mathbf{D}}, \cdot^{\mathbf{D}})$ consists of a nonempty (*abstract*) *domain* $\Delta^{\mathcal{I}}$ disjoint from $\Delta^{\mathbf{D}}$, and a mapping $\cdot^{\mathcal{I}}$ that assigns to each atomic concept $\phi \in \mathbf{A}$ a subset of $\Delta^{\mathcal{I}}$, to each individual $o \in \mathbf{I}$ an element of $\Delta^{\mathcal{I}}$, to each abstract role $R \in \mathbf{R}_A$ a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and to each datatype

role $U \in \mathbf{R}_D$ a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathbf{D}}$. We extend $\cdot^{\mathcal{I}}$ to all concepts and roles, and we define the *satisfaction* of an axiom F in an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, denoted $\mathcal{I} \models F$, as usual [16]. We say $\mathcal{I}$ *satisfies* the axiom F, or $\mathcal{I}$ is a *model* of F, iff $\mathcal{I} \models F$. We say $\mathcal{I}$ *satisfies* a knowledge base L, or $\mathcal{I}$ is a *model* of L, denoted $\mathcal{I} \models L$, iff $\mathcal{I} \models F$ for all $F \in L$. We say L is *satisfiable* iff L has a model. An axiom F is a *logical consequence* of L, denoted $L \models F$, iff every model of L satisfies F.

4 Tightly Integrated Disjunctive DL-Programs

In this section, we recall the *tightly integrated* approach to *disjunctive description logic programs* (or simply *disjunctive dl-programs*) $KB = (L, P)$ under the answer set semantics from [18], where KB consists of a description logic knowledge base L and a disjunctive logic program P. Their semantics is defined in a modular way as in [6,7], but it allows for a much tighter integration of L and P. Note that we do not assume any structural separation between the vocabularies of L and P. The main idea behind their semantics is to interpret P relative to Herbrand interpretations that are compatible with L, while L is interpreted relative to general interpretations over a first-order domain. Thus, we modularly combine the standard semantics of logic programs and of description logics, which allows for building on the standard techniques and results of both areas. As another advantage, the novel disjunctive dl-programs are decidable, even when their components of logic programs and description logic knowledge bases are both very expressive. We refer especially to [18] for further details on the novel approach to disjunctive dl-programs and for a detailed comparison to related works.

Syntax. We assume a first-order vocabulary Φ with finite nonempty sets of constant and predicate symbols, but no function symbols. We use Φ_c to denote the set of all constant symbols in Φ. We also assume a set of data values $\mathbf{V}$ (relative to a datatype theory $\mathbf{D} = (\Delta^{\mathbf{D}}, \cdot^{\mathbf{D}})$) and pairwise disjoint (denumerable) sets $\mathbf{A}$, $\mathbf{R}_A$, $\mathbf{R}_D$, and $\mathbf{I}$ of atomic concepts, abstract roles, datatype roles, and individuals, respectively, as in Section 3. We assume that (i) Φ_c is a subset of $\mathbf{I} \cup \mathbf{V}$, and that (ii) Φ and $\mathbf{A}$ (resp., $\mathbf{R}_A \cup \mathbf{R}_D$) may have unary (resp., binary) predicate symbols in common.

Let $\mathcal{X}$ be a set of variables. A *term* is either a variable from $\mathcal{X}$ or a constant symbol from Φ. An *atom* is of the form $p(t_1, \ldots, t_n)$, where p is a predicate symbol of arity $n \geqslant 0$ from Φ, and $t_1, \ldots, t_n$ are terms. A *literal* l is an atom p or a default-negated atom $not\ p$. A *disjunctive rule* (or simply *rule*) r is an expression of the form

$$\alpha_1 \vee \cdots \vee \alpha_k \leftarrow \beta_1, \ldots, \beta_n, not\ \beta_{n+1}, \ldots, not\ \beta_{n+m}\,, \tag{1}$$

where $\alpha_1, \ldots, \alpha_k, \beta_1, \ldots, \beta_{n+m}$ are atoms and $k, m, n \geqslant 0$. We call $\alpha_1 \vee \cdots \vee \alpha_k$ the *head* of r, while the conjunction $\beta_1, \ldots, \beta_n, not\ \beta_{n+1}, \ldots, not\ \beta_{n+m}$ is its *body*. We define $H(r) = \{\alpha_1, \ldots, \alpha_k\}$ and $B(r) = B^+(r) \cup B^-(r)$, where $B^+(r) = \{\beta_1, \ldots, \beta_n\}$ and $B^-(r) = \{\beta_{n+1}, \ldots, \beta_{n+m}\}$. A *disjunctive program* P is a finite set of disjunctive rules of the form (1). We say P is *positive* iff $m = 0$ for all disjunctive rules (1) in P. We say P is a *normal program* iff $k \leqslant 1$ for all disjunctive rules (1) in P.

A *tightly integrated disjunctive description logic program* (or simply *disjunctive dl-program*) $KB = (L, P)$ consists of a description logic knowledge base L and a disjunctive program P. We say KB is *positive* iff P is positive. We say KB is a *normal dl-program* iff P is a normal program.

Example 4.1. Consider the disjunctive dl-program $KB = (L, P)$, where L is the description logic knowledge base from Example 3.1, and P is the following set of rules, which express that (1) $bill$ is either a master student or a Ph.D. student (which is encoded by a rule that has the form of a disjunction of ground atoms), (2) the relation of propaedeuticity enjoys the transitive property, (3) if a student has given an exam, then he/she has given all exams that are propaedeutic to it, and (4) $unix$ is propaedeutic for $java$, and $java$ is propaedeutic for $programming\_languages$:

(1) $master\_student(bill) \vee phd\_student(bill)$;
(2) $propaedeutic(X, Z) \leftarrow propaedeutic(X, Y),\ propaedeutic(Y, Z)$;
(3) $given(X, Z) \leftarrow given(X, Y),\ propaedeutic(Z, Y)$;
(4) $propaedeutic(unix, java);\ propaedeutic(java, programming\_languages)$.

The above disjunctive dl-program also shows the advantages and flexibility of the tight integration between rules and ontologies (compared to the loose integration in [6,7]): Observe that the predicate symbol $given$ in P is also a role in L, and it freely occurs in both rule bodies and rule heads in P (which is both not possible in [6,7]). Moreover, we can easily use L to express additional constraints on the predicate symbols in P. For example, we may use the two axioms $\geqslant 1\, propaedeutic \sqsubseteq exam$ and $\geqslant 1\ propaedeutic^{-1} \sqsubseteq exam$ in L to express that $propaedeutic$ in P relates only exams.

Semantics. We now define the answer set semantics of disjunctive dl-programs as a generalization of the answer set semantics of ordinary disjunctive logic programs. In the sequel, let $KB = (L, P)$ be a disjunctive dl-program.

A *ground instance* of a rule $r \in P$ is obtained from r by replacing every variable that occurs in r by a constant symbol from Φ_c. We denote by $ground(P)$ the set of all ground instances of rules in P. The *Herbrand base* relative to Φ, denoted HB_Φ, is the set of all ground atoms constructed with constant and predicate symbols from Φ. We use DL_Φ to denote the set of all ground atoms in HB_Φ that are constructed from atomic concepts in $\mathbf{A}$, abstract roles in $\mathbf{R}_A$, and datatype roles in $\mathbf{R}_D$.

An *interpretation* I is any subset of HB_Φ. Informally, every such I represents the Herbrand interpretation in which all $a \in I$ (resp., $a \in HB_\Phi - I$) are true (resp., false). We say an interpretation I is a *model* of a description logic knowledge base L, denoted $I \models L$, iff $L \cup I \cup \{\neg a \mid a \in HB_\Phi - I\}$ is satisfiable. We say I is a *model* of a ground atom $a \in HB_\Phi$, or I *satisfies* a, denoted $I \models a$, iff $a \in I$. We say I is a *model* of a ground rule r, denoted $I \models r$, iff $I \models \alpha$ for some $\alpha \in H(r)$ whenever $I \models B(r)$, that is, $I \models \beta$ for all $\beta \in B^+(r)$ and $I \not\models \beta$ for all $\beta \in B^-(r)$. We say I is a *model* of a set of rules P iff $I \models r$ for every $r \in ground(P)$. We say I is a *model* of a disjunctive dl-program $KB = (L, P)$, denoted $I \models KB$, iff I is a model of both L and P.

We now define the answer set semantics of disjunctive dl-programs by generalizing the ordinary answer set semantics of disjunctive logic programs. We generalize the

definition via the FLP-reduct [12], which is equivalent to the standard definition via the Gelfond-Lifschitz reduct [14]. Given a dl-program $KB = (L, P)$, the *FLP-reduct* of KB relative to $I \subseteq HB_\Phi$, denoted KB^I, is the disjunctive dl-program (L, P^I), where P^I is the set of all $r \in ground(P)$ with $I \models B(r)$. Note that the *Gelfond-Lifschitz reduct* of KB relative to $I \subseteq HB_\Phi$ is the positive disjunctive dl-program $(L, \hat{P}^I)$, where $\hat{P}^I$ is obtained from $ground(P)$ by (i) deleting every rule r such that $I \models \beta$ for some $\beta \in B^-(r)$ and (ii) deleting the negative body from each remaining rule. An interpretation $I \subseteq HB_\Phi$ is an *answer set* of KB iff I is a minimal model of KB^I. A dl-program KB is *consistent* (resp., *inconsistent*) iff it has an (resp., no) answer set.

We finally define the notion of *cautious* (resp., *brave*) *reasoning* from disjunctive dl-programs under the answer set semantics as follows. A ground atom $a \in HB_\Phi$ is a *cautious* (resp., *brave*) *consequence* of a disjunctive dl-program KB under the answer set semantics iff every (resp., some) answer set of KB satisfies a.

Semantic Properties. We now summarize some important semantic properties of disjunctive dl-programs under the above answer set semantics. In the ordinary case, every answer set of a disjunctive program P is also a minimal model of P, and the converse holds when P is positive. This result holds also for disjunctive dl-programs.

As another important semantic property, the answer set semantics of disjunctive dl-programs faithfully extends its ordinary counterpart. That is, the answer set semantics of a disjunctive dl-program with empty description logic knowledge base coincides with the ordinary answer set semantics of its disjunctive program.

Furthermore, the answer set semantics of disjunctive dl-programs also faithfully extends (from the perspective of answer set programming) the first-order semantics of description logic knowledge bases. That is, a ground atom $\alpha \in HB_\Phi$ is true in all answer sets of a positive disjunctive dl-program $KB = (L, P)$ iff α is true in all first-order models of $L \cup ground(P)$. In particular, a ground atom $\alpha \in HB_\Phi$ is true in all answer sets of $KB = (L, \emptyset)$ iff α is true in all first-order models of L. Note that this result holds also when α is a ground formula constructed from HB_Φ using the operators $\wedge$ and $\vee$.

The tight integration of ontologies and rules semantically behaves very differently from the loose integration. This makes the former more (and the latter less) suitable for representing ontology mappings. The following example illustrates this difference.

Example 4.2. The normal dl-program $KB = (L, P)$, where

$$\begin{array}{rcl} L & = & \{\mathit{person}(a),\ \mathit{person} \sqsubseteq \mathit{male} \sqcup \mathit{female}\} \text{ and} \\ P & = & \{\mathit{client}(X) \leftarrow \mathit{male}(X),\ \mathit{client}(X) \leftarrow \mathit{female}(X)\} \end{array}$$

implies $\mathit{client}(a)$, while the normal dl-program $KB' = (L', P')$ as in [6,7]

$$\begin{array}{rcl} L' & = & \{\mathit{person}(a),\ \mathit{person} \sqsubseteq \mathit{male} \sqcup \mathit{female}\} \text{ and} \\ P' & = & \{\mathit{client}(X) \leftarrow \mathit{DL}[\mathit{male}](X),\ \mathit{client}(X) \leftarrow \mathit{DL}[\mathit{female}](X)\} \end{array}$$

does *not* imply $\mathit{client}(a)$, since the two queries are evaluated independently from each other, and neither $\mathit{male}(a)$ nor $\mathit{female}(a)$ follows from L'. To obtain the conclusion $\mathit{client}(a)$ in [6,7], one has to directly use the rule $\mathit{client}(X) \leftarrow \mathit{DL}[\mathit{male} \sqcup \mathit{female}](X)$.

5 Representing Ontology Mappings

In this section, we show how tightly integrated disjunctive dl-programs $KB = (L, P)$ can be used for representing (possibly inconsistent) mappings (without confidence values) between two ontologies. Intuitively, L encodes the union of the two ontologies, while P encodes the mappings between the ontologies, where disjunctions in rule heads and nonmonotonic negations in rule bodies in P can be used to resolve inconsistencies.

Tightly integrated disjunctive dl-programs $KB = (L, P)$ naturally represent two heterogeneous ontologies O_1 and O_2, and mappings between O_1 and O_2 as follows. The description logic knowledge base L is the union of two independent description logic knowledge bases L_1 and L_2, which encode the ontologies O_1 and O_2, respectively. Here, we assume that L_1 and L_2 have signatures $\mathbf{A}_1$, $\mathbf{R}_{A,1}$, $\mathbf{R}_{D,1}$, $\mathbf{I}_1$ and $\mathbf{A}_2$, $\mathbf{R}_{A,2}$, $\mathbf{R}_{D,2}$, $\mathbf{I}_2$, respectively, such that $\mathbf{A}_1 \cap \mathbf{A}_2 = \emptyset$, $\mathbf{R}_{A,1} \cap \mathbf{R}_{A,2} = \emptyset$, $\mathbf{R}_{D,1} \cap \mathbf{R}_{D,2} = \emptyset$, and $\mathbf{I}_1 \cap \mathbf{I}_2 = \emptyset$. Note that this can easily be achieved for any pair of ontologies by a suitable renaming. A mapping between elements e_1 and e_2 from L_1 and L_2, respectively, is then represented by a simple rule $e_2(\mathbf{x}) \leftarrow e_1(\mathbf{x})$ in P, where $e_1 \in \mathbf{A}_1 \cup \mathbf{R}_{A,1} \cup \mathbf{R}_{D,1}$, $e_2 \in \mathbf{A}_2 \cup \mathbf{R}_{A,2} \cup \mathbf{R}_{D,2}$, and $\mathbf{x}$ is a suitable variable vector. Informally, such a rule encodes that every instance of (the concept or role) e_1 in O_1 is also an instance of (the concept or role) e_2 in O_2. Note that demanding the signatures of L_1 and L_2 to be disjoint guarantees that the rule base that represents mappings between different ontologies is stratified as long as there are no cyclic mappings.

Example 5.1. Taking an example from the conference data set of the OAEI challenge 2006, we find e.g. the following mappings that have been created by the hmatch system for mapping the CRS Ontology (O_1) on the EKAW Ontology (O_2):

$$EarlyRegisteredParticipant(X) \leftarrow Participant(X)\,;$$
$$LateRegisteredParticipant(X) \leftarrow Participant(X)\,.$$

Informally, these two mapping relationships express that every instance of the concept $Participant$ of the ontology O_1 is also an instance of the concepts $EarlyRegistered$-$Participant$ and $LateRegisteredParticipant$, respectively, of the ontology O_2.

We now encode the two ontologies and the mappings by a tightly integrated disjunctive dl-program $KB = (L, P)$, where L is the union of two description logic knowledge bases L_1 and L_2 encoding the ontologies O_1 resp. O_2, and P encodes the mappings. However, we cannot directly use the two mapping relationships as two rules in P, since this would introduce an inconsistency in KB. More specifically, recall that a model of KB has to satisfy both L and P. Here, the two mapping relationships interpreted as rules in P would require that if there is a participant Alice ($Participant(alice)$) in the ontology O_1, an answer set of KB contains $EarlyRegisteredParticipant(alice)$ and $LateRegisteredParticipant(alice)$ at the same time. Such an answer set, however, is invalidated by the ontology O_2, which requires the concepts $EarlyRegistered$-$Participant$ and $LateRegisteredParticipant$ to be disjoint. Therefore, these mappings are useless, since they do not actively participate in the creation of any model of KB.

In [23], we present a method for detecting such inconsistent mappings. There are different approaches for resolving this inconsistency. The most straightforward one is to drop mappings until no inconsistency is present any more. Peng and Xu [25] have

proposed a more suitable method for dealing with inconsistencies in terms of a relaxation of the mappings. In particular, they propose to replace a number of conflicting mappings by a single mapping that includes a disjunction of the conflicting concepts. In the example above, we would replace the two mapping rules by the following one:

$$EarlyRegisteredParticipant(X) \vee LateRegisteredParticipant(X) \leftarrow Participant(X)\,.$$

This new mapping rule can be represented in our framework and resolves the inconsistency. More specifically, for a particular participant Alice ($Participant(alice)$) in the ontology O_1, it imposes the existence of two answer sets

$$\{EarlyRegisteredParticipant(alice), Participant(alice)\}\,;$$
$$\{LateRegisteredParticipant(alice), Participant(alice)\}\,.$$

None of these answer sets is invalidated by the disjointness constraints imposed by the ontology O_2. However, we can deduce only $Participant(alice)$ cautiously, the other atoms can be deduced bravely. More generally, with such rules, instances that are only available in the ontology O_1 cannot be classified with certainty.

We can solve this issue by refining the rules again and making use of nonmonotonic negation. In particular, we can extend the body of the original mappings with the following additional requirement:

$$EarlyRegisteredParticipant(X) \leftarrow Participant(X) \wedge RegisterdbeforeDeadline(X)\,;$$
$$LateRegisteredParticipant(X) \leftarrow Participant(X) \wedge not\ RegisteredbeforeDeadline(X)\,.$$

This refinement of the mapping rules resolves the inconsistency and also provides a more correct mapping because background information has been added. A drawback of this approach is the fact that it requires manual post-processing of mappings because the additional background information is not obvious. In the next section, we present a probabilistic extension of tightly integrated disjunctive dl-programs that allows us to directly use confidence estimations of matching engines to resolve inconsistencies and to combine the results of different matchers.

6 Tightly Integrated Probabilistic DL-Programs

In this section, we present a *tightly integrated* approach to *probabilistic disjunctive description logic programs* (or simply *probabilistic dl-programs*) under the answer set semantics. Differently from [20] (in addition to being a tightly integrated approach), the probabilistic dl-programs here also allow for disjunctions in rule heads. Similarly to the probabilistic dl-programs in [20], they are defined as a combination of dl-programs with Poole's ICL [26], but using the tightly integrated disjunctive dl-programs of [18] (see Section 4), rather than the loosely integrated dl-programs of [6,7]. Poole's ICL is based on ordinary acyclic logic programs P under different "choices", where every choice along with P produces a first-order model, and one then obtains a probability distribution over the set of all first-order models by placing a probability distribution over the different choices. We use the tightly integrated disjunctive dl-programs under the answer set semantics of [18], instead of ordinary acyclic logic programs under their canonical semantics (which coincides with their answer set semantics). We first introduce the syntax of probabilistic dl-programs and then their answer set semantics.

Syntax. We now define the syntax of probabilistic dl-programs and probabilistic queries to them. We first introduce choice spaces and probabilities on choice spaces.

A *choice space* C is a set of pairwise disjoint and nonempty sets $A \subseteq HB_{\Phi} - DL_{\Phi}$. Any $A \in C$ is an *alternative* of C and any element $a \in A$ an *atomic choice* of C. Intuitively, every alternative $A \in C$ represents a random variable and every atomic choice $a \in A$ one of its possible values. A *total choice* of C is a set $B \subseteq HB_{\Phi}$ such that $|B \cap A| = 1$ for all $A \in C$ (and thus $|B| = |C|$). Intuitively, every total choice B of C represents an assignment of values to all the random variables. A *probability* μ on a choice space C is a probability function on the set of all total choices of C. Intuitively, every probability μ is a probability distribution over the set of all variable assignments. Since C and all its alternatives are finite, μ can be defined by (i) a mapping $\mu\colon \bigcup C \rightarrow [0,1]$ such that $\sum_{a \in A} \mu(a) = 1$ for all $A \in C$, and (ii) $\mu(B) = \Pi_{b \in B} \mu(b)$ for all total choices B of C. Intuitively, (i) defines a probability over the values of each random variable of C, and (ii) assumes independence between the random variables.

A *tightly integrated probabilistic disjunctive description logic program* (or simply *probabilistic dl-program*) $KB = (L, P, C, \mu)$ consists of a disjunctive dl-program (L, P), a choice space C such that no atomic choice in C coincides with the head of any rule in $ground(P)$, and a probability μ on C. Intuitively, since the total choices of C select subsets of P, and μ is a probability distribution on the total choices of C, every probabilistic dl-program is the compact representation of a probability distribution on a finite set of disjunctive dl-programs. Observe here that P is fully general and not necessarily stratified or acyclic. We say KB is *normal* iff P is normal. A *probabilistic query* to KB has the form $\exists(c_1(\boldsymbol{x}) \vee \cdots \vee c_n(\boldsymbol{x}))[r, s]$, where $\boldsymbol{x}, r, s$ is a tuple of variables, $n \geqslant 1$, and each $c_i(\boldsymbol{x})$ is a conjunction of atoms constructed from predicate and constant symbols in Φ and variables in $\boldsymbol{x}$. Note that the above probabilistic queries can also be easily extended to conditional expressions as in [20].

Example 6.1. Consider $KB = (L, P, C, \mu)$, where L and P are as in Examples 3.1 and 4.1, respectively, except that the following two (probabilistic) rules are added to P:

$$\begin{array}{rcl} given(X, operating\_systems) & \leftarrow & master\_student(X), given(X, unix), choice_m\,; \\ given(X, operating\_systems) & \leftarrow & bachelor\_student(X), given(X, unix), choice_b\,. \end{array}$$

Let $C = \{\{choice_m, not\_choice_m\}, \{choice_b, not\_choice_b\}\}$, and let the probability μ on C be given by $\mu\colon\ choice_m,\ not\_choice_m,\ choice_b,\ not\_choice_b \mapsto 0.9,\ 0.1,\ 0.7,\ 0.3$. Here, the new (probabilistic) rules express that if a master (resp., bachelor) student has given the exam $unix$, then there is a probability of 0.9 (resp., 0.7) that he/she has also given $operating\_systems$. Note that probabilistic facts can be encoded by rules with only atomic choices in their body. Our wondering about the entailed tight interval for the probability that $john$ has given an exam on $java$ can be expressed by the probabilistic query $\exists(given(john, java))[R, S]$. Our wondering about which exams $john$ has given with which tight probability interval can be encoded by $\exists(given(john, E))[R, S]$.

Semantics. We now define an answer set semantics of probabilistic dl-programs, and we introduce the notions of consistency, consequence, tight consequence, and correct and tight answers for probabilistic queries to probabilistic dl-programs. Note that the semantics is based on subjective probabilities defined on a set of possible worlds.

Given a probabilistic dl-program $KB = (L, P, C, \mu)$, a *probabilistic interpretation* Pr is a probability function on the set of all $I \subseteq HB_\Phi$. We say Pr is an *answer set* of KB iff (i) every interpretation $I \subseteq HB_\Phi$ with $Pr(I) > 0$ is an answer set of $(L, P \cup \{p \leftarrow \mid p \in B\})$ for some total choice B of C, and (ii) $Pr(\bigwedge_{p \in B} p) = \sum_{I \subseteq HB_\Phi,\, B \subseteq I} Pr(I) = \mu(B)$ for every total choice B of C. Informally, Pr is an answer set of $KB = (L, P, C, \mu)$ iff (i) every interpretation $I \subseteq HB_\Phi$ of positive probability under Pr is an answer set of the dl-program (L, P) under some total choice B of C, and (ii) Pr coincides with μ on the total choices B of C. We say KB is *consistent* iff it has an answer set Pr.

We define the notions of consequence and tight consequence as follows. Given a probabilistic query $\exists(q(\boldsymbol{x}))[r, s]$, the *probability* of $q(\boldsymbol{x})$ in a probabilistic interpretation Pr under a variable assignment σ, denoted $Pr_\sigma(q(\boldsymbol{x}))$ is defined as the sum of all $Pr(I)$ such that $I \subseteq HB_\Phi$ and $I \models_\sigma q(\boldsymbol{x})$. We say $(q(\boldsymbol{x}))[l, u]$ (where $l, u \in [0, 1]$) is a *consequence* of KB, denoted $KB \mid\!\sim (q(\boldsymbol{x}))[l, u]$, iff $Pr_\sigma(q(\boldsymbol{x})) \in [l, u]$ for every answer set Pr of KB and every variable assignment σ. We say $(q(\boldsymbol{x}))[l, u]$ (where $l, u \in [0, 1]$) is a *tight consequence* of KB, denoted $KB \mid\!\sim_{tight} (q(\boldsymbol{x}))[l, u]$, iff l (resp., u) is the infimum (resp., supremum) of $Pr_\sigma(q(\boldsymbol{x}))$ subject to all answer sets Pr of KB and all σ. A *correct* (resp., *tight*) *answer* to a probabilistic query $\exists(c_1(\boldsymbol{x}) \vee \cdots \vee c_n(\boldsymbol{x}))[r, s]$ is a ground substitution θ (for the variables $\boldsymbol{x}, r, s$) such that $(c_1(\boldsymbol{x}) \vee \cdots \vee c_n(\boldsymbol{x}))[r, s]\,\theta$ is a consequence (resp., tight consequence) of KB.

Example 6.2. Consider again $KB = (L, P, C, \mu)$ of Example 6.1. The tight answer for $\exists(given(john, java))[R, S]$ to KB is given by $\theta = \{R/1,\ S/1\}$, while some tight answers for $\exists(given(john, E))[R, S]$ to KB are given by $\theta = \{E/java,\ R/1,\ S/1\}$, $\theta = \{E/unix,\ R/1,\ S/1\}$ and $\theta = \{E/operating\_systems,\ R/0.9,\ S/0.9\}$.

7 Representing Ontology Mappings with Confidence Values

We now show how tightly integrated probabilistic dl-programs $KB = (L, P, C, \mu)$ can be used for representing (possibly inconsistent) mappings with confidence values between two ontologies. Intuitively, L encodes the union of the two ontologies, while P, C, and μ encode the mappings between the ontologies, where confidence values can be encoded as error probabilities, and inconsistencies can also be resolved via trust probabilities (in addition to using disjunctions and nonmonotonic negations in P).

The probabilistic extension of tightly integrated disjunctive dl-programs $KB = (L, P)$ to tightly integrated probabilistic dl-programs $KB' = (L, P, C, \mu)$ provides us with a means to explicitly represent and use the confidence values provided by matching systems. In particular, we can interpret the confidence value as an *error probability* and state that the probability that a mapping introduces an error is $1 - n$. Conversely, the probability that a mapping correctly describes the semantic relation between elements of the different ontologies is $1 - (1 - n) = n$. This means that we can use the confidence value n as a probability for the correctness of a mapping. The indirect formulation is chosen, because it allows us to combine the results of different matchers in a meaningful way. In particular, if we assume that the error probabilities of two matchers are independent, we can calculate the joint error probability of two matchers that have found the same mapping rule as $(1 - n_1) \cdot (1 - n_2)$. This means that we can

get a new probability for the correctness of the rule found by two matchers which is $1-(1-n_1)\cdot(1-n_2)$. This way of calculating the joint probability meets the intuition that a mapping is more likely to be correct if it has been discovered by more than one matcher because $1-(1-n_1)\cdot(1-n_2) \geqslant n_1$ and $1-(1-n_1)\cdot(1-n_2) \geqslant n_2$.

In addition, when merging inconsistent results of different matching systems, we weigh each matching system and its result with a (user-defined) *trust probability*, which describes our confidence in its quality. All these trust probabilities sum up to 1. For example, the trust probabilities of the matching systems m_1, m_2, and m_3 may be 0.6, 0.3, and 0.1, respectively. That is, we trust most in m_1, medium in m_2, and less in m_3.

Example 7.1. We illustrate this approach using an example from the benchmark data set of the OAEI 2006 campaign. In particular, we consider the case where the publication ontology in test 101 (O_1) is mapped on the ontology of test 302 (O_2). Below we show some mappings that have been detected by the matching system hmatch that participated in the challenge. The mappings are described as rules in P, which contain a conjunct indicating the matching system that has created it and a number for identifying the mapping. These additional conjuncts are atomic choices of the choice space C and link probabilities (which are specified in the probability μ on the choice space C) to the rules (where the common concept $Proceedings$ of both ontologies O_1 and O_2 is renamed to the concepts $Proceedings_1$ and $Proceedings_2$, respectively):

$$Book(X) \leftarrow Collection(X) \wedge hmatch_1\,;$$
$$Proceedings_2(X) \leftarrow Proceedings_1(X) \wedge hmatch_2\,.$$

We define the choice space according to the interpretation of confidence described above. The resulting choice space is $C = \{\{hmatch_i, not\_hmatch_i\} \mid i \in \{1,2\}\}$. It comes along with the probability μ on C, which assigns the corresponding confidence value n (from the matching system) to each atomic choice $hmatch_i$ and the complement $1-n$ to the atomic choice not_hmatch_i. In our case, we have $\mu(hmatch_1) = 0.62$, $\mu(not\_hmatch_1) = 0.38$, $\mu(hmatch_2) = 0.73$, and $\mu(not\_hmatch_2) = 0.27$.

The benefits of this explicit treatment of uncertainty becomes clear when we now try to merge this mapping with the result of another matching system. Below are two examples of rules that describe correspondences for the same ontologies that have been found by the falcon system:

$$InCollection(X) \leftarrow Collection(X) \wedge falcon_1\,;$$
$$Proceedings_2(X) \leftarrow Proceedings_1(X) \wedge falcon_2\,.$$

Here, the confidence encoding yields the choice space $C' = \{\{falcon_i, not\_falcon_i\} \mid i \in \{1,2\}\}$ along with the probabilities $\mu'(falcon_1) = 0.94$, $\mu'(not\_falcon_1) = 0.06$, $\mu'(falcon_2) = 0.96$, and $\mu'(not\_falcon_2) = 0.04$.

Note that directly merging these two mappings as they are would not be a good idea for two reasons. The first one is that we might encounter an inconsistency problem like shown in Section 5. For example, in this case, the ontology O_2 imposes that the concepts $InCollection$ and $Book$ are to be disjoint. Thus, for each publication pub belonging to the concept $Collection$ in the ontology O_1, the merged mappings infer $Book(pub)$ and $InCollection(pub)$. Therefore, the first rule of each of the mappings cannot contribute

to a model of the knowledge base. The second reason is that a simple merge does not account for the fact that the mapping between the $Proceedings_1$ and $Proceedings_2$ concepts has been found by both matchers and should therefore be strengthened. Here, the mapping rule has the same status as any other rule in the mapping and each instance of Proceedings has two probabilities at the same time.

Suppose we associate with hmatch and falcon the trust probabilities 0.55 and 0.45, respectively. Based on the interpretation of confidence values as error probabilities, and on the use of trust probabilities when resolving inconsistencies between rules, we can now define a merged mapping set that consists of the following rules:

$$
\begin{array}{l}
Book(X) \leftarrow Collection(X) \wedge hmatch_1 \wedge sel\_hmatch_1 \,; \\
InCollection(X) \leftarrow Collection(X) \wedge falcon_1 \wedge sel\_falcon_1 \,; \\
Proceedings_2(X) \leftarrow Proceedings_1(X) \wedge hmatch_2 \,; \\
Proceedings_2(X) \leftarrow Proceedings_1(X) \wedge falcon_2 \,.
\end{array}
$$

The new choice space C'' and the new probability μ'' on C'' are obtained from $C \cup C'$ and $\mu \cdot \mu'$ (which is the product of μ and μ', that is, $(\mu \cdot \mu')(B \cup B') = \mu(B) \cdot \mu'(B')$ for all total choices B of C and B' of C'), respectively, by adding the alternative $\{sel\_hmatch_1, sel\_falcon_1\}$ and the two probabilities $\mu''(sel\_hmatch_1) = 0.55$ and $\mu''(sel\_falcon_1) = 0.45$ for resolving the inconsistency between the first two rules.

It is not difficult to verify that, due to the independent combination of alternatives, the last two rules encode that the rule $Proceedings_2(X) \leftarrow Proceedings_1(X)$ holds with the probability $1 - (1 - \mu''(hmatch_2)) \cdot (1 - \mu''(falcon_2)) = 0.9892$, as desired. Informally, any randomly chosen instance of $Proceedings$ of the ontology O_1 is also an instance of $Proceedings$ of the ontology O_2 with the probability 0.9892. In contrast, if the mapping rule would have been discovered only by falcon or hmatch, respectively, such an instance of $Proceedings$ of the ontology O_1 would be an instance of $Proceedings$ of the ontology O_2 with the probability 0.96 or 0.73, respectively.

A probabilistic query Q asking for the probability that a specific publication pub in the ontology O_1 is an instance of the concept $Book$ of the ontology O_2 is given by $Q = \exists(Book(pub))[R, S]$. The tight answer θ to Q is given by $\theta = \{R/0, S/0\}$, if pub is not an instance of the concept $Collection$ in the ontology O_1 (since there is no mapping rule that maps another concept than $Collection$ to the concept $Book$). If pub is an instance of the concept $Collection$, however, then the tight answer to Q is given by $\theta = \{R/0.341, S/0.341\}$ (as $\mu''(hmatch_1) \cdot \mu''(sel\_hmatch_1) = 0.62 \cdot 0.55 = 0.341$). Informally, pub belongs to the concept $Book$ with the probabilities 0 resp. 0.341. Note that we may obtain real intervals when there are total choices with multiple answer sets.

8 Algorithms and Complexity

In this section, we characterize the consistency and the query processing problem in probabilistic dl-programs in terms of the consistency and the cautious/brave reasoning problem in disjunctive dl-programs (which are all decidable [18]). These characterizations show that the consistency and the query processing problem in probabilistic dl-programs are decidable resp. computable, and they directly reveal algorithms for solving these problems. We also give a precise picture of the complexity of deciding consistency and correct answers when the choice space C is bounded by a constant.

Algorithms. The following theorem shows that a probabilistic dl-program $KB = (L, P, C, \mu)$ is consistent iff $(L, P \cup \{p \leftarrow \mid p \in B\})$ is consistent, for every total choice B of C with $\mu(B) > 0$. This implies that deciding whether a probabilistic dl-program is consistent can be reduced to deciding whether a disjunctive dl-program is consistent.

Theorem 8.1. *A probabilistic dl-program $KB = (L, P, C, \mu)$ is consistent iff $(L, P \cup \{p \leftarrow \mid p \in B\})$ is consistent for every total choice B of C with $\mu(B) > 0$.*

The next theorem shows that computing tight answers for $\exists(q)[r, s]$ to KB, where $q \in HB_\Phi$, can be reduced to brave and cautious reasoning from disjunctive dl-programs. Informally, to obtain the tight lower (resp., upper) bound, we have to sum up all $\mu(B)$ such that q is a cautious (resp., brave) consequence of $(L, P \cup \{p \leftarrow \mid p \in B\})$. The theorem holds also when q is a ground formula constructed from HB_Φ. Note that this result implies also that tight query processing in probabilistic dl-programs KB can be done by an anytime algorithm (along the total choices of KB).

Theorem 8.2. *Let $KB = (L, P, C, \mu)$ be a consistent probabilistic dl-program, and let q be a ground atom from HB_Φ. Then, l (resp., u) such that $KB \| \sim_{tight} (q)[l, u]$ is the sum of all $\mu(B)$ such that (i) B is a total choice of C and (ii) q is true in all (resp., some) answer sets of $(L, P \cup \{p \leftarrow \mid p \in B\})$.*

Complexity. The following theorem shows that deciding whether a probabilistic dl-program is consistent is complete for $\mathrm{NEXP}^{\mathrm{NP}}$ (and so has the same complexity as deciding consistency in ordinary disjunctive logic programs) when the size of its choice space is bounded by a constant. Here, the lower bound follows from the $\mathrm{NEXP}^{\mathrm{NP}}$-hardness of deciding whether an ordinary disjunctive logic program has an answer set [5].

Theorem 8.3. *Given Φ and a probabilistic dl-program $KB = (L, P, C, \mu)$, where L is defined in $\mathcal{SHIF}(\mathbf{D})$ or $\mathcal{SHOIN}(\mathbf{D})$, and the size of C is bounded by a constant, deciding whether KB is consistent is complete for* $\mathrm{NEXP}^{\mathrm{NP}}$.

The next theorem shows that deciding correct answers for probabilistic queries $\exists(q)[r, s]$, where $q \in HB_\Phi$, to a probabilistic dl-program is complete for co-$\mathrm{NEXP}^{\mathrm{NP}}$ when the size of the choice space is bounded by a constant. The theorem holds also when q is a ground formula constructed from HB_Φ.

Theorem 8.4. *Given Φ, a probabilistic dl-program $KB = (L, P, C, \mu)$, where L is defined in $\mathcal{SHIF}(\mathbf{D})$ or $\mathcal{SHOIN}(\mathbf{D})$, and the size of C is bounded by a constant, a ground atom q from HB_Φ, and $l, u \in [0, 1]$, deciding whether $(q)[l, u]$ is a consequence of KB is complete for* co-$\mathrm{NEXP}^{\mathrm{NP}}$.

9 Tractability Results

In this section, we describe a special class of probabilistic dl-programs for which deciding consistency and query processing can both be done in polynomial time in the data complexity. These programs are normal, stratified, and defined relative to *DL-Lite* [4], which allows for deciding knowledge base satisfiability in polynomial time.

We first recall *DL-Lite*. Let $\mathbf{A}$, $\mathbf{R}_A$, and $\mathbf{I}$ be pairwise disjoint sets of atomic concepts, abstract roles, and individuals, respectively. A *basic concept in DL-Lite* is either an atomic concept from $\mathbf{A}$ or an exists restriction on roles $\exists R.\top$ (abbreviated as $\exists R$), where $R \in \mathbf{R}_A \cup \mathbf{R}_A^-$. A *literal in DL-Lite* is either a basic concept b or the negation of a basic concept $\neg b$. *Concepts in DL-Lite* are defined by induction as follows. Every basic concept in *DL-Lite* is a concept in *DL-Lite*. If b is a basic concept in *DL-Lite*, and ϕ_1 and ϕ_2 are concepts in *DL-Lite*, then $\neg b$ and $\phi_1 \sqcap \phi_2$ are also concepts in *DL-Lite*. An *axiom in DL-Lite* is either (1) a concept inclusion axiom $b \sqsubseteq \phi$, where b is a basic concept in *DL-Lite*, and ϕ is a concept in *DL-Lite*, or (2) a *functionality axiom* (funct R), where $R \in \mathbf{R}_A \cup \mathbf{R}_A^-$, or (3) a concept membership axiom $b(a)$, where b is a basic concept in *DL-Lite* and $a \in \mathbf{I}$, or (4) a role membership axiom $R(a, c)$, where $R \in \mathbf{R}_A$ and $a, c \in \mathbf{I}$. A *knowledge base in DL-Lite* L is a finite set of axioms in *DL-Lite*.

Every knowledge base in *DL-Lite* L can be transformed into an equivalent one in *DL-Lite* $trans(L)$ in which every concept inclusion axiom is of form $b \sqsubseteq \ell$, where b (resp., ℓ) is a basic concept (resp., literal) in *DL-Lite* [4]. We then define $trans(P) = P \cup \{b'(X) \leftarrow b(X) \mid b \sqsubseteq b' \in trans(L), b' \text{ is a basic concept}\} \cup \{\exists R(X) \leftarrow R(X, Y) \mid R \in \mathbf{R}_A \cap \Phi\} \cup \{\exists R^-(Y) \leftarrow R(X, Y) \mid R \in \mathbf{R}_A \cap \Phi\}$. Intuitively, we make explicit all the relationships between the predicates in P that are implicitly encoded in L.

We define stratified normal dl- and stratified normal probabilistic dl-programs as follows. A normal dl-program $KB = (L, P)$ is *stratified* iff (i) L is defined in *DL-Lite* and (ii) $trans(P)$ is locally stratified. A probabilistic dl-program $KB = (L, P, C, \mu)$ is *normal* iff P is normal. A normal probabilistic dl-program $KB = (L, P, C, \mu)$ is *stratified* iff every of KB's represented dl-programs is stratified.

The following result shows that stratified normal probabilistic dl-programs allow for consistency checking and query processing with a polynomial data complexity. It follows from Theorems 8.1 and 8.2 and that consistency checking and reasoning in stratified normal dl-programs can be done in polynomial time in the data complexity [18].

Theorem 9.1. *Given Φ and a stratified normal probabilistic dl-program KB, (a) deciding if KB has an answer set, and (b) computing $l, u \in [0, 1]$ for a given ground atom q such that $KB \mid\!\sim_{tight}(q)[l, u]$ can be done in polynomial time in the data complexity.*

10 Conclusion

We have presented tightly integrated probabilistic (disjunctive) dl-programs as a rule-based framework for representing ontology mappings that supports the resolution of inconsistencies on a symbolic and a numeric level. While the use of disjunction and nonmonotonic negation allows the rewriting of inconsistent rules, the probabilistic extension of the language allows us to explicitly represent numeric confidence values as error probabilities, to resolve inconsistencies by using trust probabilities, and to reason about these on a numeric level. While being expressive and well-integrated with description logic ontologies, the language is still decidable and has data-tractable subsets that make it particularly interesting for practical applications.

Note that probabilistic queries in tightly integrated probabilistic dl-programs can syntactically and semantically easily be generalized to contain conditionals of disjunctions of conjunctions of atoms, rather than only disjunctions of conjunctions of atoms.

The characterization in Theorem 8.2 can be generalized to such probabilistic queries, and the completeness for co-$\mathrm{NEXP}^{\mathrm{NP}}$ of Theorem 8.4 also carries over to them. Furthermore, note that tightly integrated probabilistic dl-programs are also a natural approach to combining languages for reasoning about actions with both description logics and Bayesian uncertainty (which is especially directed towards Web Services) [1].

We leave for future work the implementation of tightly integrated probabilistic dl-programs. Another interesting topic for future work is to explore whether the tractability results can be extended to an even larger class of tightly integrated probabilistic dl-programs. One way to achieve this could be to approximate the answer set semantics through the well-founded semantics (which may be defined similarly as in [21]). Furthermore, it would be interesting to investigate whether one can develop an efficient top-k query technique (as in [28,22]) for tightly integrated probabilistic dl-programs: Rather than computing the tight probability interval for a given ground atom, such a technique returns the k most probable ground instances of a given non-ground atom.

Acknowledgments. Andrea Calì is supported by the STREP FET project TONES (FP6-7603) of the EU. Thomas Lukasiewicz is supported by the German Research Foundation (DFG) under the Heisenberg Programme and by the Austrian Science Fund (FWF) under the project P18146-N04. Heiner Stuckenschmidt and Livia Predoiu are supported by an Emmy-Noether Grant of the German Research Foundation (DFG).

Appendix A: Proofs

Proof of Theorem 8.1. Recall first that KB is consistent iff KB has an answer set Pr, which is a probabilistic interpretation Pr such that (i) every interpretation $I \subseteq HB_\Phi$ with $Pr(I) > 0$ is an answer set of the disjunctive dl-program $(L, P \cup \{p \leftarrow \mid p \in B\})$ for some total choice B of C, and (ii) $Pr(\bigwedge_{p \in B} p) = \mu(B)$ for each total choice B of C.

($\Rightarrow$) Suppose that KB is consistent. We now show that the disjunctive dl-program $(L, P \cup \{p \leftarrow \mid p \in B\})$ is consistent, for every total choice B of C with $\mu(B) > 0$. Towards a contradiction, suppose the contrary. That is, $(L, P \cup \{p \leftarrow \mid p \in B\})$ is not consistent for some total choice B of C with $\mu(B) > 0$. So, it follows that $Pr(\bigwedge_{p \in B} p) = 0$. But this contradicts $Pr(\bigwedge_{p \in B} p) = \mu(B) > 0$. This shows that $(L, P \cup \{p \leftarrow \mid p \in B\})$ is consistent, for every total choice B of C with $\mu(B) > 0$.

($\Leftarrow$) Suppose that the disjunctive dl-program $(L, P \cup \{p \leftarrow \mid p \in B\})$ is consistent, for every total choice B of C with $\mu(B) > 0$. That is, there exists some answer set I_B of $(L, P \cup \{p \leftarrow \mid p \in B\})$, for every total choice B of C with $\mu(B) > 0$. Let the probabilistic interpretation Pr be defined by $Pr(I_B) = \mu(B)$ for every total choice B of C with $\mu(B) > 0$ and by $Pr(I) = 0$ for all other $I \subseteq HB_\Phi$. Then, Pr is an interpretation that satisfies (i) and (ii). That is, Pr is an answer set of KB. Thus, KB is consistent. □

Proof of Theorem 8.2. The statement of the theorem follows from the observation that the probability $\mu(B)$ of all total choices B of C such that q is true in all (resp., some) answer sets of $(L, P \cup \{p \leftarrow \mid p \in B\})$ contributes (resp., may contribute) to the probability $Pr(q)$, while the probability $\mu(B)$ of all total choices B of C such that q is false in all answer sets of $(L, P \cup \{p \leftarrow \mid p \in B\})$ does not contribute to $Pr(q)$. □

Proof of Theorem 8.3. We first show membership in $\mathrm{NEXP}^{\mathrm{NP}}$. By Theorem 8.1, we check whether $(L, P \cup \{p \leftarrow \mid p \in B\})$ is consistent, for every total choice B of C with $\mu(B) > 0$. Since C is bounded by a constant, the number of all total choices B of C with $\mu(B) > 0$ is also bounded by a constant. As shown in [18], deciding whether a disjunctive dl-program has an answer set is in $\mathrm{NEXP}^{\mathrm{NP}}$. In summary, this shows that deciding whether KB is consistent is in $\mathrm{NEXP}^{\mathrm{NP}}$.

Hardness for $\mathrm{NEXP}^{\mathrm{NP}}$ follows from the $\mathrm{NEXP}^{\mathrm{NP}}$-hardness of deciding whether a disjunctive dl-program has an answer set [18], since by Theorem 8.1 a disjunctive dl-program $KB = (L, P)$ has an answer set iff the probabilistic dl-program $KB = (L, P, C, \mu)$ has answer set, for the choice space $C = \{\{a\}\}$, the probability function $\mu(a) = 1$, and any ground atom $a \in HB_{\Phi}$ that does not occur in $ground(P)$. □

Proof of Theorem 8.4. We first show membership in co-$\mathrm{NEXP}^{\mathrm{NP}}$. We show that deciding whether $(q)[l, u]$ is not a consequence of KB is in $\mathrm{NEXP}^{\mathrm{NP}}$. By Theorem 8.2, $(q)[l, u]$ is not a consequence of KB iff there exists a set $\mathcal{B}$ of total choices B of C such that either (a.1) q is true in some answer set of $(L, P \cup \{p \leftarrow \mid p \in B\})$, for every $B \in \mathcal{B}$, and (a.2) $\sum_{B \in \mathcal{B}} \mu(B) > u$, or (b.1) q is false in some answer set of $(L, P \cup \{p \leftarrow \mid p \in B\})$, for every $B \in \mathcal{B}$, and (a.2) $\sum_{B \in \mathcal{B}} \mu(B) < l$. As shown in [18], deciding whether q is true in some answer set of a disjunctive dl-program is in $\mathrm{NEXP}^{\mathrm{NP}}$. It thus follows that deciding whether $(q)[l, u]$ is not a consequence of KB is in $\mathrm{NEXP}^{\mathrm{NP}}$, and thus deciding whether $(q)[l, u]$ is a consequence of KB is in co-$\mathrm{NEXP}^{\mathrm{NP}}$

Hardness for co-$\mathrm{NEXP}^{\mathrm{NP}}$ follows from the co-$\mathrm{NEXP}^{\mathrm{NP}}$-hardness of deciding whether a ground atom q is true in all answer sets of a disjunctive dl-program [18], since by Theorem 8.2 a ground atom q is true in all answer sets of a disjunctive dl-program $KB = (L, P)$ iff $(q)[1, 1]$ is a consequence of the probabilistic dl-program $KB = (L, P, C, \mu)$ under the answer set semantics, for the choice space $C = \{\{a\}\}$, the probability function $\mu(a) = 1$, and any $a \in HB_{\Phi}$ that does not occur in $ground(P)$. □

Proof of Theorem 9.1. As shown in [18], deciding the existence of (and computing) the answer set of a stratified normal dl-program has a polynomial data complexity. Observe then that in the case of data complexity, the choice space C is fixed. By Theorems 8.1 and 8.2, it thus follows that the problems of (a) deciding whether KB has an answer set, and (b) computing $l, u \in [0, 1]$ for a given ground atom q such that $KB \mid\!\sim_{tight} (q)[l, u]$, respectively, can both be solved in polynomial time in the data complexity. □

References

1. A. Calì and T. Lukasiewicz. Tightly integrated probabilistic description logic programs for the Semantic Web. In *Proc. ICLP-2007*, pp. 428–429. *LNCS* 4670, Springer, 2007.
2. P. C. G. da Costa. *Bayesian Semantics for the Semantic Web*. Doctoral Dissertation, George Mason University, Fairfax, VA, USA, 2005.
3. P. C. G. da Costa and K. B. Laskey. PR-OWL: A framework for probabilistic ontologies. In *Proc. FOIS-2006*, pp. 237–249. IOS Press, 2006.
4. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. *DL-Lite*: Tractable description logics for ontologies. In *Proc. AAAI-2005*, pp. 602–607. AAAI Press / MIT Press, 2005.

5. E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.
6. T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining answer set programming with description logics for the Semantic Web. In *Proc. KR-2004*, pp. 141–151. AAAI Press, 2004.
7. T. Eiter, G. Ianni, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining answer set programming with description logics for the Semantic Web. Technical Report INFSYS RR-1843-07-04, Institut für Informationssysteme, TU Wien, March 2007.
8. T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. Effective integration of declarative rules with external evaluations for semantic-web reasoning. In *Proc. ESWC-2006*, pp. 273–287. *LNCS* 4011, Springer, 2006.
9. J. Euzenat, M. Mochol, P. Shvaiko, H. Stuckenschmidt, O. Svab, V. Svatek, W. R. van Hage, and M. Yatskevich. First results of the ontology alignment evaluation initiative 2006. In *Proc. ISWC-2006 Workshop on Ontology Matching*.
10. J. Euzenat and P. Shvaiko. *Ontology Matching*. Springer, Heidelberg, Germany, 2007.
11. J. Euzenat, H. Stuckenschmidt, and M. Yatskevich. Introduction to the ontology alignment evaluation 2005. In *Proc. K-CAP-2005 Workshop on Integrating Ontologies*.
12. W. Faber, N. Leone, and G. Pfeifer. Recursive aggregates in disjunctive logic programs: Semantics and complexity. In *Proc. JELIA-2004*, pp. 200–212. *LNCS* 3229, Springer, 2004.
13. A. Finzi and T. Lukasiewicz. Structure-based causes and explanations in the independent choice logic. In *Proc. UAI-2003*, pp. 225–232. Morgan Kaufmann, 2003.
14. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, 9(3/4):365–386, 1991.
15. R. Giugno and T. Lukasiewicz. P-$\mathcal{SHOQ}(\mathbf{D})$: A probabilistic extension of $\mathcal{SHOQ}(\mathbf{D})$ for probabilistic ontologies in the Semantic Web. In *Proc. JELIA-2002*, pp. 86–97. *LNCS* 2424, Springer, 2002.
16. I. Horrocks and P. F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In *Proc. ISWC-2003*, pp. 17–29. *LNCS* 2870, Springer, 2003.
17. I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In *Proc. LPAR-1999*, pp. 161–180. *LNCS* 1705, Springer, 1999.
18. T. Lukasiewicz. A novel combination of answer set programming with description logics for the Semantic Web. In *Proc. ESWC-2007*, pp. 384-398. *LNCS* 4519, Springer, 2007.
19. T. Lukasiewicz. Expressive probabilistic description logics. *Artif. Intell.*, in press.
20. T. Lukasiewicz. Probabilistic description logic programs. *Int. J. Approx. Reason.*, 45(2):288–307, 2007.
21. T. Lukasiewicz. Tractable probabilistic description logic programs. In *Proc. SUM-2007*, pp. 143–156. *LNCS* 4772, Springer, 2007.
22. T. Lukasiewicz and U. Straccia. Top-k retrieval in description logic programs under vagueness for the Semantic Web. In *Proc. SUM-2007*, pp. 16–30. *LNCS* 4772, Springer, 2007.
23. C. Meilicke, H. Stuckenschmidt, and A. Tamilin. Repairing ontology mappings. In *Proc. AAAI-2007*, pp. 1408–1413. AAAI Press, 2007.
24. B. Motik, I. Horrocks, R. Rosati, and U. Sattler. Can OWL and logic programming live together happily ever after? In *Proc. ISWC-2006*, pp. 501–514. *LNCS* 4273, Springer, 2006.
25. P. Wang and B. Xu. Debugging ontology mapping: A static method. *Computation and Intelligence*, 2007. To appear.
26. D. Poole. The independent choice logic for modelling multiple agents under uncertainty. *Artif. Intell.*, 94(1/2):7–56, 1997.
27. L. Serafini, H. Stuckenschmidt, and H. Wache. A formal investigation of mapping languages for terminological knowledge. In *Proc. IJCAI-2005*, pp. 576–581, 2005.
28. U. Straccia. Towards top-k query answering in description logics: The case of *DL-Lite*. In *Proc. JELIA-2006*, pp. 439–451. *LNCS* 4160, Springer, 2006.