Adding Probabilities and Rules to OWL Lite Subsets based on Probabilistic Datalog

HENRIK NOTTELemann
University of Duisburg-Essen
47048 Duisburg, Germany
nottelmann@uni-duisburg.de

NORBERT FUHR
University of Duisburg-Essen
47048 Duisburg, Germany
fuhr@uni-duisburg.de

Received (14 January 2005)
Revised (2 September 2005)
Accepted (5 October 2005)

This paper proposes two probabilistic extensions of variants of the OWL Lite description language, which are essential for advanced applications like information retrieval. The first step follows the axiomatic approach of combining description logics and Horn clauses: Subsets of OWL Lite are mapped in a sound and complete way onto Horn predicate logics (Datalog variants). Compared to earlier approaches, a larger fraction of OWL Lite can be transformed by switching to Datalog with equality in the head; however, some OWL Lite constructs cannot be transformed completely into Datalog. By using probabilistic Datalog, the new probabilistic OWL Lite subsets (both with support for Horn rules) are defined, and the semantics are given by the semantics of the corresponding probabilistic Datalog program. As inference engines for probabilistic Datalog are available, description logics and information retrieval systems can easily be combined.

Keywords: OWL Lite, probabilistic Datalog, logics, rules, information retrieval

1. Introduction

In contrast to traditional (relational) databases, unstructured or semi-structured documents (e.g. in XML) with an uncertain content representation are searched in information retrieval (IR) with respect to (w. r. t.) a vague user information need. Typically, the output is a list of documents ranked w. r. t. numeral weights. Probabilistic models become more and more popular, as optimum retrieval has only been defined precisely for them: The Probability Ranking Principle\(^{26}\) says that optimum retrieval quality is achieved if documents are ranked with decreasing probability that the document is relevant to a user. The combination of probability theory and a logical framework bears additional opportunities for information retrieval: queries, documents and retrieval approaches can be presented in a clear way, and it is straightforward to incorporate external knowledge like a thesaurus, hyper-links between documents or ontologies. Examples are PLBR (propositional logic and belief revision)\(^{18}\) as
well as probabilistic Datalog\textsuperscript{7}, which is based on Horn predicate logics with recursion and negation and allows for computing point probabilities (in contrast to intervals as for other approaches). The latter has been successfully employed for information retrieval\textsuperscript{22} and for the problem of mapping between different schemas or ontologies\textsuperscript{24}.

A new and challenging problem is information retrieval in the context of the Semantic Web, where information is structured and presented employing the OWL ontology language\textsuperscript{21,25}. This work is based on restricted versions of OWL Lite\textsuperscript{a}. Like other description logic languages, the OWL language suffers from two significant restrictions when using it for information retrieval:

(1) OWL does not support general rules yet, and the rule extension SWRL\textsuperscript{12} is still under development. Rules could be used as a powerful IR query language, for defining mappings between ontologies, or for intelligent Semantic Web Services.

(2) OWL is only able to specify and infer deterministic knowledge. Uncertain knowledge is required, however, e.g. for query answering in IR, or when wrappers are used to derive ontology instances from HTML pages (where e.g. misspelling, synonyms or page irregularities lead to some errors).

In this paper, OWL Lite is restricted to two subsets, called OWL Lite\textsuperscript{−} and OWL Lite\textsuperscript{EQ}:

\[
\emptyset \subseteq \text{OWL Lite}^{-} \subset \text{OWL Lite}^{EQ} \subset \text{OWL Lite} \subseteq \text{OWL DL} \subseteq \text{OWL Full}.
\]

Two directions can be distinguished for combining Horn logics and description logics: The DL-log approach\textsuperscript{3} allows description logics concepts and roles in Horn rule bodies only. Thus, new Datalog facts can be inferred, but the description logic knowledge base is not altered by the rules. In contrast, the axiomatic approach maps concepts and roles onto Datalog predicates. This allows for inferring new knowledge for description logic concepts and facts, but is more difficult to achieve.

This work follows the axiomatic approach. The two languages OWL Lite\textsuperscript{−} and OWL Lite\textsuperscript{EQ}, are mapped onto Datalog and Datalog\textsuperscript{EQ}, respectively in a model- and entailment-preserving way. Both OWL Lite subsets are then extended by uncertain assertions and (uncertain) rules based on the resulting probabilistic Datalog program. This allows for using pOWL Lite\textsuperscript{−} and pOWL Lite\textsuperscript{EQ} for information retrieval in actual applications. Probabilistic Datalog is chosen because inference engines are available, either on the basis of HySpirit\textsuperscript{8}, or by mapping facts and rules onto SQL statements.

The remainder of this paper is organised as follows. First we give a brief introduction into Datalog variants. Sections 3 and 4 introduce OWL Lite\textsuperscript{−} and OWL Lite\textsuperscript{EQ}, while problems with mapping the remaining OWL Lite constructs are discussed in section 5. Uncertainty and rules are added to OWL Lite in section 6. An example is given in section 7, and related work is discussed in section 8.

\textsuperscript{a}OWL Lite is strongly related to the \textit{S}\textit{H}\textit{I}\textit{F}(\textit{D}) description logics.
2. (Probabilistic) Datalog variants

This sections gives a brief overview over Datalog and its deterministic and probabilistic extensions.

2.1. Datalog

Datalog is a variant of predicate logic based on function-free Horn clauses. Terms are constants (functions without arguments) or variables (placeholders for constants); in the remainder, \(x\), \(y\) and \(z\) always denote variables, all other terms are constants. Atoms have the form \(p(\vec{t})\), where \(p\) is an \(n\)-ary predicate, \(\vec{t} = (t_1, \ldots, t_n)\) is a vector of \(n\) terms. A (Datalog) literal is an atom \(p(\vec{t})\), its negation \(\neg p(\vec{t})\), or \(t_1 = t_2\) or \(t_1 \neq t_2\).

Rules have the form \(h \leftarrow b_1 \land \cdots \land b_l\), where \(h\) ("head") is an atom and the \(b_i\) (the subgoals of the "body") denote literals. A rule can be seen as a clause \(\{h, \neg b_1, \ldots, \neg b_l\}\) or the disjunction \(\neg p(\vec{t}) \lor \neg q(\vec{t}) \lor r(\vec{t})\) or the equivalent rule \(r(\vec{t}) \leftarrow p(\vec{t}), q(\vec{t})\). Please note that Datalog does not allow (in)equality in the rule head, it can only occur in the rule body. Finally, a fact is a rule with only constants in the head and an empty body; the arrow \(\leftarrow\) can be omitted then: \(p(a_1, \ldots, a_n)\).

This example program denotes that father\((x, y)\) is true for fixed values for the variables \(x\) and \(y\) if both parent\((x, y)\) and male\((x)\) are true:

\[
\text{parent}(jo, mary), \\
\text{father}(x, y) \leftarrow \text{parent}(x, y), \text{male}(x).
\]

The semantics are based on the Herbrand universe, which equals the set of all constants defined in the program (if no constant is defined, a new constant is introduced) in the case of Datalog. The Herbrand base consists of all possible ground atoms (atoms without variables) w.r.t. all defined predicates the Herbrand universe. In the example from above, it consists of:

\[
\{\text{parent}(jo, jo), \text{parent}(jo, mary), \text{parent}(mary, jo), \text{parent}(mary, mary), \\
\text{father}(jo, jo), \text{father}(jo, mary), \text{father}(mary, jo), \text{father}(mary, mary), \\
\text{male}(jo), \text{male}(mary)\}
\]

An interpretation (also called Herbrand interpretation) then maps each member of the Herbrand universe (i.e., constants) onto itself, and each member of the Herbrand base (i.e., ground atoms) onto the truth values “true” and “false” (in an alternative but equivalent definition, a Datalog interpretation is the subset of the Herbrand base which are considered to be true). A model is an interpretation which satisfies all facts and rules. The precise semantics are defined by total well-founded models, which induces that each Datalog program has at most one minimal model (the Datalog model).

Here, we allow for modularly stratified programs only, which ensures that each Datalog program has exactly one (unique) total minimal model. Since the definition of modular stratification is rather complicated, we only give a simple explanation: In contrast to global stratification, modular stratification is formulated w.r.t. the instantiation of a program for its
Herbrand universe. The program is modularly stratified if there is an assignment of ordinal levels to ground atoms such that whenever a ground atom appears negatively in the body of a rule, the ground atom in the head of that rule is of strictly higher level, and whenever a ground atom appears positively in the body of a rule, the ground atom in the head has at least that level. Informally, in modularly stratified programs, no ground atom can depend negatively upon itself.

### 2.2. Datalog\(^\text{EQ}\)

Datalog\(^\text{EQ}\), a simple Datalog extension, allows for equality in the rule heads as in:

\[
x = y \leftarrow b_1(t), \ldots, b_l(t).
\]

The body of such a rule is a standard Datalog rule body and must contain both variables occurring in the head. The idea is that the constants denoted by the variables \(x\) and \(y\) can be used interchangeably, without modifying the meaning of the program. In other words, they are defined to be in the same equivalence class.

Datalog is based on the Herbrand universe, where every interpretation \(I = (\mathcal{U}, \cdot^I)\) maps each constant onto itself: \(\forall x \in \mathcal{U} : x^I = x\). In Datalog\(^\text{EQ}\), each interpretation maps each member of the Herbrand universe onto another member of the Herbrand universe, but the mapping is not injective (i.e., several constants can be mapped onto the same other constant). A minimal model, then, also requires that the sets of symbols not mapped onto themselves are minimal; this property ensures that all Datalog\(^\text{EQ}\) models are equivalent.

Informally, each symbol is mapped onto itself unless two constants are inferred to be equal by a rule; the representative of the constants can be chosen arbitrarily.

For instance, each person has only one father, but maybe the fact basis uses different names (e.g., first and middle name) for the same person:

\[
x = y \leftarrow \text{father}(x, z), \text{father}(y, z).
\]

Datalog\(^\text{EQ}\) can be reduced to a standard Datalog program with an equivalent unique minimal model. Let \(I = (\mathcal{U}, I^I)\) be the model of the Datalog\(^\text{EQ}\) program (with interpretation function \(I\)), and \(J = (\mathcal{U}, J^I)\) be the model of the resulting Datalog program (with \(x^I = x\)), and \(r\) an \(n\)-ary predicate:

\[
\forall x_1, \ldots, x_n \in \mathcal{U} : r(x_1^I, \ldots, x_n^I) I^I \iff r(x_1^I, \ldots, x_n^I) J^I.
\]

The reduction process works as follows. First, a new binary predicate \(\text{eq}\) is introduced. For each constant \(a\) occurring in the Datalog\(^\text{EQ}\) program, the fact \(\text{eq}(a, a)\) is added to the resulting Datalog program. The predicate \(\text{eq}\) is symmetric and transitive, which is modelled by the rules:

\[
\text{eq}(y, x) \leftarrow \text{eq}(x, y).
\]

\[
\text{eq}(x, z) \leftarrow \text{eq}(x, y), \text{eq}(y, z).
\]
Furthermore, $n$ additional recursive rules are introduced for each $n$-ary predicate, dealing with equivalence classes:

\[
p(x_1, \ldots, x_n) \leftarrow p(x'_1, \ldots, x'_n), \text{eq}(x_1, x'_1) \\
\vdots \\
p(x_1, \ldots, x_n) \leftarrow p(x_1, \ldots, x'_n), \text{eq}(x_n, x'_n)
\]

Literals $x = y$ in the rule head and the rule body are replaced by the literal $\text{eq}(x, y)$. Literals $x \neq y$ (which are only allowed to appear in the rule body) are replaced by $\neg \text{eq}(x, y)$.

### 2.3. Datalog\textsuperscript{IC} and Datalog\textsuperscript{IC,EQ}

Datalog\textsuperscript{IC} introduces integrity constraints to Datalog. These constraints are rules with empty heads, which means that the rule bodies are evaluated to “false” by every model.

We have two options for mapping integrity constraints onto standard Datalog. For both options, all constraints are removed, and the model of the resulting Datalog program is computed. In a second step, the model is dropped if any constraint is violated (i.e., it is evaluated to “false”).

The easiest way would be to replace empty rule heads by the ground atom $\text{status}(\text{inconsistent})$. If the model of the resulting program contains the fact $\text{status}(\text{inconsistent})$, it has been inferred by a constraint. Thus, there is an inconsistency, and the program is unsatisfiable (i.e., the model has to be dropped).

Alternatively, instead of requiring obscure facts for flagging inconsistencies, we can switch to a language which is able to express inconsistency in models: four-valued Datalog with an additional closed-world assumption. Four-valued Datalog\textsuperscript{b} uses—beside the truth values “true” and “false” from two-valued Datalog—also “unknown” (if we don’t have any information) and “inconsistent” (if there is evidence that it is true as well as evidence that it is not true).

Four-valued Datalog can be used for modelling Datalog\textsuperscript{IC}: In a first step, we eliminate “unknown” by assigning “false” to each ground atom which is not known to be true. In a second step, the fact $\text{status}(\text{consistent})$ is added to the program, and the literal $\neg \text{status}(\text{consistent})$ replaces empty rule heads. If the constraint is violated, then the model contains both $\text{status}(\text{consistent})$ and $\neg \text{status}(\text{consistent})$, which means the corresponding fact has the truth value “inconsistent”.

Datalog\textsuperscript{IC,EQ} is the combination of Datalog\textsuperscript{EQ} and Datalog\textsuperscript{IC}.

### 2.4. Probabilistic Datalog

In (two-valued) probabilistic Datalog\textsuperscript{7} (pDatalog for short), every fact or rule has a probabilistic weight $\alpha \in [0, 1]$ attached, prepended to the fact or rule:

\[
\alpha h(\bar{t}) \leftarrow b_1(\bar{t}), \ldots, b_l(\bar{t}).
\]

\textsuperscript{b}Four-valued Datalog has been defined in the context of its probabilistic variant\textsuperscript{8}. 


A weight $\alpha = 1$ can be omitted. In that case the rule is called deterministic. Each fact and rule can only appear once in the program, to avoid inconsistencies. The intended meaning of a rule $\alpha r$ is that “for any instantiated rule $r$, the probability that it is true is $\alpha$.”

The following program expresses that a person is with probability of 50% male, that “Ed” is a person with probability of 80%:

\[
0.5 \text{male}(x) \leftarrow \text{person}(x).
\]
\[
0.8 \text{person}(\text{ed}).
\]

By default, any two probabilistic events are assumed to be independent, so the probabilities of the conjunction of two events can be multiplied, i.e. we yield $\Pr(\text{male}(\text{ed})) = 0.8 \cdot 0.5 = 0.4$ for our example. This independence assumption allows for computing point probabilities, and is quite popular in the field of information retrieval for which pDatalog has originally been developed.

Computation of the probabilities is based on the notion of event keys and event expressions, which allow for recognising duplicate or disjoint events. Facts and instantiated expressions, which allow for recognising duplicate or disjoint events.

Formally, an interpretation structure in pDatalog is a tuple $\mathcal{I} = (W, \mu)$, where $W$ is a set of possible worlds and $\mu$ is a probability distribution over $W$. The possible worlds are defined as follows. Given a pDatalog program $P$, with $H(P)$ we indicate the ground instantiation of $P$. Then, the deterministic part of $P$ is the set $P_D$ of instantiated rules in $H(P)$ having weight $\alpha = 1$, while the indeterministic part of $P$ is the set $P_I$ of instantiated rules determined by $P_I = \{ r : \alpha r \in H(P), \alpha < 1 \}$. The set of deterministic programs of $P$, denoted $D(P)$ is defined as $D(P) = \{ P_I \cup Y : Y \subseteq P_I \}$. Note that any $P' \in D(P)$ is a classical logic program. Finally, a possible world $w \in W$ is the minimal model $\mathcal{I}$ of a deterministic program in $D(P)$. Now, an interpretation is a tuple $I = (\mathcal{I}, w)$ such that $w \in W$. The notion of truth w. r. t. an interpretation and a possible world can be defined recursively:

\[
(\mathcal{I}, w) \models A \text{ iff } A \in w,
\]
\[
(\mathcal{I}, w) \models A \leftarrow B_1, \ldots, B_n \text{ iff } (\mathcal{I}, w) \models B_1, \ldots, B_n \Rightarrow (\mathcal{I}, w) \models A,
\]
\[
(\mathcal{I}, w) \models \alpha r \text{ iff } \mu(\{ w' \in W : (\mathcal{I}, w') \models r \}) = \alpha.
\]

An interpretation $(\mathcal{I}, w)$ is a model of a pDatalog program $P$, denoted $(\mathcal{I}, w) \models P$, iff it entails every fact and rule in $P$:

\[
(\mathcal{I}, w) \models P \text{ iff } (\mathcal{I}, w) \models \alpha r, \text{ for all } \alpha r \in H(P).
\]

3. Mapping OWL Lite$^-$ onto Datalog

Bruijn et.al.² introduced OWL Lite$^-$ as a subset of OWL Lite which can be transformed into an equivalent Datalog program: Classes and properties are mapped onto unary and
binary Datalog predicates, assertions onto facts, and axioms and restrictions onto Datalog rules.

OWL Lite− has been designed so that it can be transformed into an equivalent Datalog program. While OWL Lite itself can have multiple minimal models, each OWL Lite− program has one unique minimal model, which is equivalent to the (unique minimal) model of the corresponding Datalog program. For this reason, the transformation process is model preserving, and thus, also entailment preserving for named concepts (details are described in section 3.4). Only a proper subset of OWL Lite (i.e., the subset OWL Lite−) can be mapped, this issue is discussed in section 5.

For simplicity, constants like "2003-12-23" are enclosed in quotation marks in Datalog.

We will use two notations simultaneously, the OWL Abstract Syntax and a description logics syntax (remind that OWL Lite is closely related to the SHI (D) description logics). In the latter, A and B are named concepts, R and S are (named) roles, a and b are individuals, and x, y and z are variables. In the presentation, we merely use the description logics terms “concept” and “role” rather than the OWL synonyms “class” and “property”. Both notations are used throughout the remainder of this paper, along with a textual description. For a precise definition of OWL, the reader is referred to the OWL Semantics and Abstract Syntax documentation.

3.1. Definitions and assertions

First, concepts and roles have to be defined in OWL Lite−. In addition, one can specify that two individuals are the same (to bridge the gap between heterogeneous ontologies over the same domain), or that two individuals are different. Finally, individuals can be declared to be instances of concepts, and pairs of individuals to be instances of roles (assertions).

Class (A): Named concepts (called classes in OWL) are defined as sets of objects, and can be mapped onto unary Datalog predicates with the same name. In Datalog, we do not have to formally define predicates and their arities, as this is clear from their first usage in facts and rules. Restrictions are discussed below.

Property (R): Similar to concepts, roles (properties in OWL, i.e. sets of pairs an objects and another object or a fixed value) are mapped onto binary Datalog predicates with the same name.

Individual(…) (a, b): Individuals are encoded as Datalog constants. All OWL individuals occurring in the knowledge base are added to the unary Datalog predicate U, which will later ensure that some rules are safe (i.e., all variables in the rule heads occur in positive subgoals in the rule body):

\[ a, b \implies U(a) \land U(b). \]

\(^4\)If no individual is defined, a dummy individual is introduced to ensure a non-empty universe.
OWL Lite adheres to the unique name assumption: Individuals with different “names” (URIs) are considered to be different. This implies that it is not possible to state that two different URIs name the same individual. This adheres to Datalog, which is based on the usage of the Herbrand universe, where any interpretation maps each constant onto itself.

\[ a \in B \implies B(a). \]
\[ (a, b) \in R \implies R(a, b). \]

In addition, it is also possible to state: If \((a, b)\) is contained in the role \(R\), then \(b\) must be an instance of the named concept \(B\). This assertion corresponds to the universal quantifier restriction (see section 3.2).

\[ a \in \forall R.B \implies B(y) \leftarrow R(a, y). \]  

### 3.2. Concept axioms and universal quantifier restriction

Concept axioms are used to define a hierarchy of concepts. Thus, OWL Lite allows to state that a concept is the sub-set of the conjunction of other concepts (named concepts or a special form of an OWL restrictions), or that it is equivalent to the conjunction of other concepts.

\[ A \sqsubseteq B_1 \sqcap \ldots \sqcap B_n \implies \]
\[ B_1(x) \leftarrow A(x) \]
\[ \vdots \]
\[ B_n(x) \leftarrow A(x) \]

Note that \(B_i = \top\) is allowed but useless (\(A \sqsubseteq \top\) holds anyway), and the corresponding rule will be removed. For \(A = \top\), the rule bodies are replaced by \(\mathcal{U}(x)\) (the universe).

In addition, \(A = \bot\) is allowed but useless, as \(\bot \sqsubseteq B_i\) holds anyway; the corresponding rule will simply be removed. However, \(B_i = \bot\) would yield \(A \sqsubseteq \bot\), which refers to an integrity constraint, and thus cannot be expressed in Datalog.

\[ A \equiv B_1 \sqcap \ldots \sqcap B_n \implies \]
\[ A(x) \leftarrow B_1(x), \ldots, B_n(x). \]

Concerning \(\top\) and \(\bot\), similar arguments as for the partial definition hold: \(\top\) is allowed on both sides of the complete class definition, while \(\bot\) is not allowed at all.
Class(…partial restriction(…allValuesFrom…))) (A ⊑ ∀R.B): This construct (the universal quantifier restriction) states that if (a,b) is contained in the role R and a is an instance of A, that then b must be an instance of the named concept B:

\[ A \sqsubseteq \forall R.B \implies B(y) \leftarrow A(x), R(x,y). \] (3)

Here, A = ⊤ is allowed (simplifying the rule by removing A(x) from the rule body), and A = ⊥ is legal but useless, and drops the rule at all. On the other hand, we impose B \neq ⊥, as B = ⊥ would be equivalent to an integrity constraint. In addition, B = ⊤ yields A = ⊤, so the rule has to be changed as described above.

Note that only \sqsubseteq is allowed (i.e., only partial restrictions), \equiv is not supported in OWL Lite−.

3.3. Role axioms

Axioms are also possible for roles.

ObjectProperty(…super …) or SubPropertyOf (R ⊑ S): Role hierarchies are created by this feature. Similar to the concept hierarchy, this can be transformed into the simple rule:

\[ R \sqsubseteq S \implies S(x,y) \leftarrow R(x,y) \]

EquivalentProperties (R \equiv S): This can be transformed into two rules:

\[ R \equiv S \implies S(x,y) \leftarrow R(x,y) \]
\[ R(x,y) \leftarrow S(x,y) \]

inverseOf (R = S−): This construct, stating that the role R = S− is the inverse of S, can be expressed in Datalog as:

\[ R = S^{-} \implies R(x,y) \leftarrow S(y,x) \]
\[ S(x,y) \leftarrow R(y,x) \]

Symmetric (R = R−): A role may be stated to be symmetric. This is equivalent to specifying that a role is inverse to itself:

\[ R = R^{-} \implies R(x,y) \leftarrow R(y,x) \]

Transitive (R+ ⊑ R): For a role R, R+ defines the transitive closure. As a consequence, a role can be defined as transitive by:

\[ R^{+} \sqsubseteq R \implies R(x,z) \leftarrow R(x,y), R(y,z) \]
The fact that named concept $B$ is domain of the role $R$ can be easily expressed as:

$$\top \sqsubseteq \forall R \cdot B \implies B(x) \leftarrow R(x, y)$$

The fact that named concept $B$ is range of the role $R$ can be easily expressed as:

$$\top \sqsubseteq \forall R \cdot B \implies B(y) \leftarrow R(x, y)$$

3.4. Soundness and completeness of the transformation process

OWL Lite$^-$ is defined by the constructs (and only the constructs) discussed in sections 3.1-3.3. The semantics of OWL Lite$^-$ coincide with the semantics of OWL Lite on the supported language fragment. OWL Lite$^-$ is designed so that each knowledge base has one unique minimal model: It does not provide disjunction, negation (which enforces monotonicity) and equality constraints, and, thus, the usage of the universal quantifier is restricted, and the existential quantifier and cardinality restrictions are disallowed.

We further showed how OWL Lite$^-$ can be mapped onto Datalog. The minimal model of knowledge base $KB$ is equivalent to the minimal model of the corresponding Datalog program $P$:

$$KB \models a \in A \iff P \models A(a),$$

$$KB \models (a, b) \in R \iff P \models R(a, b).$$

for individuals/constants $a$ and $b$, a named concept/unary predicate $A$ and a role/binary predicate $R$. This property can be proven directly based on the mappings, as the semantics of all OWL/Datalog pairs involved in the mappings are equivalent.

Thus, the mapping is sound and complete: A model of $P$ is also a model of $KB$, and vice versa.

4. Mapping OWL Lite$^E$ onto Datalog$^E$

In contrast to plain Datalog, Datalog$^E$ allows for equality in the head, which specifies that two constants should be treated as the same (i.e., they are mapped onto the same value in every interpretation). The penalty is a lower efficiency, as recursive rules using the $eq$ predicate are added.

OWL Lite$^E$ is defined by mapping its constructs onto Datalog$^E$. This section describes the features in OWL Lite$^E \setminus$ OWL Lite$^-$ (the other constructs have already been discussed in section 3).

4.1. Definitions and assertions

OWL Lite$^-$, similarly to Datalog, is based on a unique name assumption, i.e. different URIs (constants in Datalog) name different individuals. In OWL Lite, however, the same
individuals can have multiple “names” (i.e., URIs). Thus, it is possible to explicitly state that two individuals are the same or different:

\[
\text{sameIndividualAs}(a = b): \text{This will be covered by the rule:} \\
\begin{align*}
  a = b \quad \implies \quad a = b \gets U(a), U(b).
\end{align*}
\]

\[
\text{differentIndividualFrom}(a \neq b): \text{In Datalog}^{\text{EQ}}, \text{all constants are different by default, so this directive does not need any mapping.}
\]

4.2. Cardinality restrictions
One of the important cardinality constraints can easily be mapped onto Datalog\textsuperscript{EQ}:

\[
\text{maxCardinality(1)}(\leq 1R. \top): \text{This construct states that two fillers for role } R \text{ are actually the same. This construct can be transformed into:}
\begin{align*}
  &\text{diff2R}(x) \gets R(x, y), R(x, z), y \neq z \\
  &a \subseteq 1R. \top \quad \implies \quad y = z \gets R(a, y), R(a, z) \\
  &A \subseteq 1R. \top \quad \implies \quad y = z \gets A(x), R(x, y), R(x, z) \\
  &A \subseteq 1R. \top \quad \implies \quad A(x) \gets U(x), \neg \text{diff2R}(x)
\end{align*}
\]

4.3. Role axioms
The (inverse) functional role axioms cannot be mapped onto Datalog, as they require equality in the rule head. However, they can be easily included into OWL Lite\textsuperscript{EQ}:

\[
\text{Functional}(\top \subseteq 1R. \top): \text{This construct states that a role is functional, i.e. it has at most one filler } y \text{ with } R(x, y) \text{ for an individual } x. \text{ So, from } R(x, y) \text{ and } R(x, z) \text{ we can conclude } y = z. \text{ This behaviour can be expressed as:}
\begin{align*}
  \top \subseteq 1R. \top \quad \implies \quad y = z \gets R(x, y), R(x, z).
\end{align*}
\]

\[
\text{InverseFunctional}(\top \subseteq 1R^{-}. \top): \text{This construct states that the inverse of a role is functional. As above, this can be expressed in Datalog:}
\begin{align*}
  \top \subseteq 1R^{-}. \top \quad \implies \quad x = y \gets R(x, z), R(y, z).
\end{align*}
\]

4.4. Soundness and completeness of the transformation process
As for OWL Lite\textsuperscript{-}, the mapping of OWL Lite\textsuperscript{EQ} onto Datalog\textsuperscript{EQ} directly models the semantics of OWL Lite in Horn rules, thus also this mapping is sound and complete.
5. From OWL Lite\textsuperscript{EQ} to OWL Lite

In sections 3 and 4, only subsets of OWL Lite (OWL Lite\textsuperscript{−} and OWL Lite\textsuperscript{EQ}, respectively) have been mapped onto Datalog, which yields a complete and sound transformation. We will now show that this subset has been chosen carefully, as the missing fragments (which are not included in OWL Lite\textsuperscript{−} and OWL Lite\textsuperscript{EQ}) cannot be mapped onto Datalog\textsuperscript{EQ}.

5.1. Quantifier restrictions

Quantifier restrictions are used as anonymous concepts which can be used as super-concepts or equivalent concepts. In OWL Lite\textsuperscript{−}, only the universal quantifier $\forall R.B$ is supported as a super-concept, all other quantifier restriction constructs are dropped from the language.

By switching to Datalog\textsuperscript{IC}, however, we are able to detect “inconsistencies”, i.e. knowledge bases which are unsatisfiable. We will see later, however, that the resulting Datalog\textsuperscript{IC} program has no model (as a constraint is violated) although the corresponding description logic knowledge base is satisfiable.

Even though we still miss some inferences, this approach is superior to approaches like OWL Lite\textsuperscript{−} or the work by Grosof et.al.\textsuperscript{10} as these constraints can be handled at least partially, and consistent Datalog programs imply that also the description logic knowledge base is consistent.

\textbf{Class}(\ldots\text{complete restriction}(...allValuesFrom(...))) (A $\sqsubseteq \forall R.B$): This states that every instance whose $R$ role fillers are all included in $B$ are also instances of $A$, where $A$ and $B$ are named concepts.

A naive mapping, utilising the equivalence $\forall R.B = \neg\exists R.\neg B$, would be:

\begin{align*}
\text{existsRnotB}(x) & \leftarrow R(x, y), \neg B(y) \\
A \sqsupseteq \forall R.B & \implies A(x) \leftarrow \forall U(x), \neg \text{existsRnotB}(x).
\end{align*}

Unfortunately, this mapping directly leads to Datalog programs which are not modularly stratified: As A $\sqsupseteq \forall R.B$ can only be used together with A $\sqsubseteq \forall R.B$, we have a dependency cycle in Datalog (using $A$ and $B$) through negation: The predicate $A$ depends negatively upon \text{existsRnotB}, which depends upon $B$, which itself depends upon $A$.\textsuperscript{d} Thus, the program cannot be evaluated.

As an alternative approach, we switch to Datalog\textsuperscript{IC} and detect inconsistencies. In addition, we introduce a new predicate $frB$ as a subset of the concept $\forall R.B$ which allows partial inference (see below):

\begin{align*}
\text{existsRnotB}(x) & \leftarrow R(x, y), \neg B(y) \\
A \sqsupseteq \forall R.B & \implies \forall U(x), \neg A(x), \neg \text{existsRnotB}(x) \\
A(x) & \leftarrow frB(x).
\end{align*}

\textsuperscript{d}Modular stratification depends on grounded rules and not on the rules alone. However, one can easily see that the argument holds when a fact $R(a, a)$ is given.
Adding Probabilities and Rules to OWL Lite Subsets based on Probabilistic Datalog

\[ B(y) \leftarrow f_{RB}(x), R(x, y) \]

Please note that we do not introduce rules which infer new facts for \( f_{RB} \), as these rules would suffer from the same problem (Datalog programs which are not modularly stratified) as the naive mapping rule (4). However, we can infer knowledge about \( f_{RB} \) from the opposite direction (equations (2) and (3)): We replace the OWL Lite \( \sqsubseteq \forall R.B \) by these ones which infer facts for \( f_{BR} \):

\[
\begin{align*}
A \sqsubseteq \forall R.B & \implies B(y) \leftarrow A(x), R(x, y).
\quad \text{(5)} \\
\forall R.B \sqsubseteq & \implies B(y) \leftarrow f_{RB}(x) \leftarrow A(x).
\quad \text{(6)}
\end{align*}
\]

Together we are e.g. able to infer \( a \in C \) given the example rules:

\[
\begin{align*}
A \sqsubseteq \forall R.B \\
\forall R.B \sqsubseteq C \\
a \in A, \ b \in B \\
(c, b) \in R
\end{align*}
\]

In addition, in OWL Lite we would be able to infer \( c \in C \), but the Datalog program does not allow for this kind of inference.

Special rules have to be built for \( \bot \) and \( \top \):

\[
\begin{align*}
\exists \exists R (x) & \leftarrow R(x, y)
\end{align*}
\]

However, nothing is ever inferred from \( \exists \exists R \neg B(x) \), as it only occurs in constraints.

**someValuesFrom** \((\exists R.B)\): This construct (the existential quantifier) states that if \( a \) is an instance of \( A \), then there must be at least one instance \( b \) of the named concept \( B \) so that \((a, b)\) is contained in the role \( R \). This construct can only partially be transformed into
Datalog. The existential quantifier refers to infinite disjunction, which cannot be expressed in Datalog rule heads. Thus, we can check for inconsistencies (if there is an instance of $A$ which does not have a filler for $R$) employing Datalog IC:

$$\exists RB(x) \leftarrow R(x,y), B(y)$$

$$a \in \exists R.B$$ $$\implies$$ $$\neg \exists RB(a)$$

$$A \sqsubseteq \exists R.B$$ $$\implies$$ $$\neg A(x), \neg \exists RB(x)$$

$$A \sqsupseteq \exists R.B$$ $$\implies$$ $$A(x) \leftarrow \exists RB(x)$$

For the assertion and the $\sqsubseteq$ direction (equation (9)), we are only able to detect inconsistencies, as it is not possible to express the existential quantification in a rule head. The other direction (equation (10)) is easier, as it directly refers to Datalog rules.

There are cases, however, where the knowledge base is satisfiable but the Datalog IC program has no model (as at least one constraint is violated): Consider the knowledge base

$$a \in \exists R.B$$

$$b \in B$$

and the resulting program:

$$\exists RB(x) \leftarrow R(x,y), B(y)$$

$$\neg \exists RB(a)$$

$$B(b).$$

One of the models of this knowledge base is $\{B(b), R(a,b)\}$; however, the last fact cannot be inferred in Datalog IC (the constraints are applied on the computed Datalog model), and thus the program has no model.

We can augment this with the same trick as in rules (5) and (6):

$$eRB(x) \leftarrow \exists RB(x).$$

$$A \sqsubseteq \exists R.B$$ $$\implies$$ $$eRB(x) \leftarrow A(x).$$

$$A \sqsupseteq \exists R.B$$ $$\implies$$ $$A(x) \leftarrow eRB(x)$$

As before, this is not an equivalent mapping, but preserves some (although not all) inferences.

Special rules have to be built for $\bot$ and $\top$:

$$\exists RS(x) \leftarrow R(x,y)$$

$$A \sqsubseteq \exists R, \bot$$ $$\implies$$ $$\neg A(x)$$

$$A \sqsupseteq \exists R, \bot$$ $$\implies$$

$$A \sqsubseteq \exists R, \top$$ $$\implies$$ $$\neg A(x), \neg \exists RS(x)$$

$$A \sqsupseteq \exists R, \top$$ $$\implies$$ $$A(x) \leftarrow \exists RS(x)$$

$$\bot \sqsubseteq \exists R, B$$ $$\implies$$

$$\bot \sqsupseteq \exists R, B$$ $$\implies$$ $$\neg \exists RS(x)$$
5.2. Cardinality restrictions

Like quantifier restrictions, cardinality restrictions are used as anonymous concepts which can be used as super-concepts or equivalent concepts in the form \( A \sqsubseteq C \) or \( A \equiv C \) for a named concept \( A \) and a restriction \( C \). Here, \( A \sqsubseteq C \) has to be transformed into an integrity constraint, which can be expressed in Datalog\(^{IC}\) but not in Datalog.

- **minCardinality(0) (\( \geq 0 \). \( \top \))**: With \( \geq 0 \). \( \top \) = \( \top \), this constraint is useless.

- **minCardinality(1) (\( \geq 1 \). \( \top \))**: With \( \geq 1 \). \( \top \) = \( \exists \). \( \top \), we have a similar situation as for the existential quantifier (see section 5.1).

- **maxCardinality(0) (\( \leq 0 \). \( \top \))**: This construct states that the role \( R \) is forbidden for every instance of concept \( A \):

\[
\exists R. B \quad \Rightarrow \quad \neg \exists R B(x)
\]

\[
\exists R. B \quad \Rightarrow \quad \neg \exists R B(x)
\]

Again, the first direction (equation (11)) can only be transformed into an integrity constraint to detect inconsistencies. This can be augmented with the same trick (introducing a new predicate \( \text{max}0R \)) as for the quantifier restrictions (equations (5) and 6).

- **cardinality(0,1) (\( = 0 \). \( \top \), \( = 1 \). \( \top \))**: This is just a shortcut, and does not need any further mapping.

5.3. Soundness and completeness of the transformation process

Mapping OWL Lite onto Datalog\(^{IC,EO}\) is not model-preserving: OWL Lite knowledge bases can have multiple minimal models, whereas Datalog and, with it, Datalog\(^{IC,EO}\), has one unique minimal model.

The mapping from OWL Lite onto Datalog\(^{IC,EO}\) is not complete: We have shown for each construct in OWL Lite \( \setminus \) OWL Lite\(^{EO}\) that we preserve some inferences, but miss others.

The mapping is, however, sound: A model of the Datalog\(^{IC,EO}\) program is also a model of the knowledge base. For each construct, we followed the OWL Lite semantics, and created rules which do not infer facts which we cannot infer from the knowledge base (actually, most mappings create constraints, which do not infer new facts). Since we restrict to the monotonic part of OWL Lite\(^{−}\), also the mapping of a complete knowledge base is sound (although it is not complete).
6. Adding probabilities and rules to OWL Lite− and OWL LiteEQ

So far, we demonstrated how we can map knowledge bases using variants of OWL Lite onto Datalog programs. Facts are used for OWL assertions, and Datalog rules for modelling the OWL semantics of OWL concept and role axioms.

OWL Lite and Datalog allow for deterministic (Boolean) facts only. Often, however, we need to incorporate uncertainty. A simple example would be that it is uncertain if a man is really the parent of another person. In addition, advanced applications require support for general rules, e.g. for ontology mappings, information retrieval or planning agents.

By switching to probabilistic Datalog, we extend OWL Lite− and OWL LiteEQ in three directions:

1. uncertain assertions,
2. uncertain axioms (some of them), and
3. probabilistic Horn rules.

The OWL Lite variant names are then prefixed with a “p”. The language pOWL Lite− is derived by extending OWL Lite− with probabilities and rules as described in sections 6.1-6.4, and pOWL LiteEQ equals OWL Lite− with probabilities and rules. Semantics of pOWL Lite− and pOWL LiteEQ are defined by the corresponding Datalog (and DatalogEQ) program.

Note that a subset of the Horn rule mechanism does not require probabilistic Datalog. We decided, however, to keep OWL Lite− and OWL LiteEQ as subsets of OWL, and thus did not include Horn rule logic support into these two deterministic languages.

6.1. Probabilistic definitions and assertions

Assertions and equality definitions can be weighted in pOWL Lite− and pOWL LiteEQ:

**Individual(...type ...value(...)) (a ∈ B, (a,b) ∈ R)**: The assertion weight obviously is the probability of the corresponding pDatalog fact:

\[ \alpha a \in B \implies \alpha B(a) \]
\[ \alpha (a,b) \in R \implies \alpha R(a,b) \]
\[ \alpha a \in \forall R.B \implies \alpha B(y) \leftarrow R(a,y) \]

In the abstract syntax, the probability is prepended to the weighted construct, and can be omitted if the probability equals one:

\[ Pr(Bob \in Person) = 0.7 \]
\[ Pr((Peter, Mary) \in parent) = 0.9 \]

is expressed in the OWL abstract syntax as:
Adding Probabilities and Rules to OWL Lite Subsets based on Probabilistic Datalog

Individual(Bob 0.7 type(Person))
Individual(Peter 0.9 value(parent Mary))

This knowledge base can be mapped onto:

0.7 Person(Bob).
0.9 parent(Peter, Mary).

sameIndividualAs (a = b): It is also possible to specify a probability that two individuals are the same:

\[ \alpha a = b \implies \alpha a = b \leftarrow U(a), U(b). \]

6.2. Probabilistic concept axioms and restrictions

Weighting of axioms is allowed where there is a clear meaning for the weighted axioms which can be easily transformed into probabilistic Datalog:

\textbf{Class(\ldots partial \ldots )} (A \sqsubseteq B_1 \cap \ldots \cap B_n): This construct defines concept hierarchies. Each \( B_i \), which has to be a named concept in this case, is weighted by a probabilistic weight \( \alpha_i \in [0, 1] \):

\[ \alpha_i A_i \sqsubseteq B \implies \alpha_i B(x) \leftarrow A_i(x). \]

E.g., one can specify that a 49% of all persons are men as:

\textbf{Class} (Person partial 0.49 Man)

This is mapped onto \textbf{(CCC: disjointness)}

\[ 0.49 \text{ Man}(x) \leftarrow \text{Person}(x). \]

Thus, the probabilistic axioms do not mean “with a probability of 0.49, all persons are men”, but refer to grounded rules (where the variables are replaced by individuals), which coincides with the meaning of probabilistic Datalog rules, and the semantics of probabilistic terminological knowledge in other approaches (see section 8).

\textbf{Class(\ldots partial restriction(\ldots allValuesFrom(\ldots )))} (A \sqsubseteq \forall R.B): This construct also defines concept hierarchies using the universal quantifier restriction. It has also an instance-based interpretation:

\[ \alpha A \sqsubseteq \forall R.B \implies \alpha B(y) \leftarrow A(x), R(x, y). \]

In a concrete example,

Class (Person partial 0.5 restriction{parent allValuesFrom(Man)})

can be mapped onto

\[ 0.5 \text{ Man}(y) \leftarrow \text{Person}(x), \text{parent}(x, y). \]
Thus, the probabilistic axiom refers to grounded rules (where the variables are replaced by individuals), and states that a parent of a person is with probability of 0.5 a man.

\[ \text{maxCardinality}(1) \ (A \sqsubseteq 1 R \top) : \text{Again, we follow an instance-based interpretation} \]
\[ \text{and view the weight as the probability that two role fillers are actually the same.} \]
\[ \text{This construct can be transformed into:} \]
\[ \alpha A \sqsubseteq 1 R \top \implies \alpha y = z \leftarrow A(x), R(x, y), R(x, z) \]

### 6.3. Probabilistic role axioms

Most role axioms can be weighted, and the axiom weight is used as the weight of the corresponding Datalog rule. Only \text{EquivalentProperties} and \text{inverseOf} are excluded as they are transformed into two recursive rules (see section 3.3).

\text{ObjectProperty(...super ...) or SubPropertyOf} (R \sqsubseteq S): This case of role hierarchies is equivalent to concept hierarchies.

\text{Symmetric} (R = R^{-}): Here, the probability specifies the probability of the corresponding rule.

\[ \alpha R = R^{-} \implies \alpha R(x, y) \leftarrow R(y, x) \]

\text{Transitive} (R^{+} \sqsubseteq R): Here, the weight specifies the probability of the corresponding rule, similarly to the symmetric role case.

\text{domain} (\top \sqsubseteq \forall R^{-}.B): The probabilistic weight specifies the probability of the corresponding rule, i.e. for an individual \( a \) with \( R(a, b) \), that \( a \) is in \( B \):

\[ \alpha \top \sqsubseteq \forall R^{-}.B \implies \alpha B(x) \leftarrow R(x, y). \]

\text{range} (\top \sqsubseteq \forall R.C): This is analogous to the domain definition.

\text{Functional} (\top \sqsubseteq 1 R \top): The weight assigned to this construct specifies the probability that a the role has at most one filler \( y \) with \( R(x, y) \) for an individual \( x \):

\[ \alpha \top \sqsubseteq 1 R \top \implies \alpha y = z \leftarrow R(x, y), R(x, z). \]

\text{InverseFunctional} (\top \sqsubseteq 1 R^{-} \top): This is analogous to the functional role definition.
6.4. Horn rules

Advanced semantic applications require support for stating rules. Syntax and semantics of rules are based on the Semantic Web Rule Language (SWRL)\(^{12}\), but its use is restricted as rules are mapped onto pDatalog rules.

The basic form of a rule is:

\[
\text{Implies(\text{Antecedent}(...), \text{Consequent}(...))}
\]

Here, the consequent (the rule head) contains a single literal, while the antecedent (the rule body) contains a list of literals. A literal has the form \(p(x_1, \ldots, x_n)\) or \(\neg p(x_1, \ldots, x_n)\), where \(p\) is a named concept \(A\), a role \(R\) or any other \(n\)-ary predicate which is integrated in the reasoner or available in a pDatalog program. In the antecedent, also \(x = y\) and \(x \neq y\) are supported. In contrast to SWRL, restrictions are not allowed.

Thus, literals have one of the following forms:

- \(\text{foo(bar,...,baz)}\)
- \(\text{not foo(bar,...,baz)}\)
- \(\text{sameAs(bar,baz)}\)
- \(\text{differentFrom(bar,baz)}\)

Instead of the individuals \(\text{bar}\) and \(\text{baz}\), also variables can be used:

\(\text{foo(I-variable(x))}\).

The consequent can be weighted, where the probabilistic weight is prepended to the \(\text{Consequent}\) construct. Such a pOWL Lite rule can directly be transformed into a pDatalog rule. E.g., we can express that 49% of all persons are men as this rule instead of an axiom:

\[
\text{Implies(\text{Antecedent}(\text{Man(I-variable(x)})), 0.49 \text{Consequent(Person(I-variable(x))}))}
\]

Section 7 provides a more complex example which shows that pOWL Lite\(^{-}\) rules increase the expressiveness of the language, as such a rule cannot be expressed in OWL Lite\(^{-}\) alone.

7. Example

Uncertain knowledge bases and support for general rules are important for several applications. In the following example, we show the usefulness of pOWL Lite\(^{-}\) by combining mappings between heterogeneous ontologies, wrappers with uncertain output, classification and information retrieval on these facts in a seamless way.

The scenario is sports on TV. Ontologies describe sport teams, their locations, meteorology data, and TV shows. pOWL Lite\(^{-}\) constructs allow for specifying complex classification and retrieval tasks.
7.1. Sports teams

Our first ontology defines cities, the cities in a distance lower than 2 miles, groups of peoples and their location, and baseball teams (located in cities):

Class(Group).
ObjectProperty(in).
ObjectProperty(in2Miles Symmetric domain(City) range(City))
Class(BaseballTeam partial Group restriction(in allValuesFrom(City)))).

In an alternative ontology, only sports teams are specified, and a mapping onto the first ontology is given (assuming that 40% of all sport teams are baseball teams):

Class(SportTeam partial Group restriction(in allValuesFrom(City)))).
Class(SportTeam partial 0.4 BaseballTeam).

With the following assertions

Individual(Boston type(City) in2Miles(Cambridge)).
Individual(Cambridge type(City)).
Individual(RedSox type(SportTeam) in(Boston)).

pOWL Lite can infer:

Individual(RedSox 0.4 type(BaseballTeam)).

7.2. Meteorology

A third ontology deals with weather reports:

Class(Hot partial City).
DatatypeProperty(temp domain(City) range(xsd:integer)).
Implies(Antecedent(temp(I-variable(x),"30"))
Consequent(Hot(I-variable(x))))
Implies(Antecedent(Hot(I-variable(x)),in2Miles(I-variable(x),I-variable(y)))
0.8 Consequent(Hot(I-variable(y))))
Individual(Boston 0.8 temp("30") 0.2 temp("29")).
Individual(Cambridge 0.9 temp("30") 0.1 temp("29")).

Thus, we conclude that it is hot in a city with probability of 80% if it is hot in another city which is less than 2 miles away, or if it is hot in a city if the temperature is 30°C. The uncertain facts are due to minor deviations of measured temperatures.

The following facts can be inferred:

Individual(Cambridge 0.9 type(Hot))).
Individual(Boston 0.944 type(Hot))).

The probability of 0.944 in the last line is computed as follows: From above, a city with a temperature of 30°C is a hot city, so following this rule, Boston has hot weather with a
probability of 0.8. On the other hand, if another hot city is less than 2 miles away, than the city is hot with a probability of 0.8; as Cambridge is such close, and it is hot with a probability of 0.9, Boston is hot (following only this rule) with a probability of 0.8 · 0.9. Finally, we have to compute the disjunction of the two rules, following the inclusion-exclusion formula, deriving 0.8 + 0.8 · 0.9 − 0.8 · 0.8 · 0.9 = 0.944.

7.3. TV shows

Another ontology defines TV shows with sub-classes, and text terms recognised from the reporter’s voice:

\[
\begin{align*}
&\text{Class(Show)} \\
&\text{DatatypeProperty(text domain(Show) range(xsd:string))} \\
&\text{Class(BaseballShow partial Show)} \\
&\text{ObjectProperty(teamInShow domain(BaseballShow) range(Team))} \\
&\text{Implies(Antecedent(Show(I-variable(x)),text(I-variable(x),"pitcher"),text(I-variable(x),"bat"))} \\
&\quad 0.7 \text{ Consequent(BaseballShow(I-variable(x))))} \\
&\text{Implies(Antecedent(Show(I-variable(x)),text(I-variable(x),"weather"))} \\
&\quad 0.8 \text{ Consequent(WeatherShow(I-variable(x))))} \\
&\text{Class(WeatherShow partial Show)} \\
&\text{Class(FootballWeatherShow partial FootballShow WeatherShow)}
\end{align*}
\]

Thus, shows are classified according to the terms used by the reporter.⁶

If a user searches for baseball TV shows about the Red Sox team which mention the weather, and the weather actually was hot, she could state this rule in an information retrieval task:

\[
\begin{align*}
&\text{Implies(Antecedent(BaseballShow(I-variable(x)),} \\
&\quad \text{teamInShow(I-variable(x),RedSox),} \\
&\quad \text{in(I-variable(x),I-variable(y)),} \\
&\quad \text{Hot(I-variable(y))})} \\
&\quad \text{Consequent(about(x))}).
\end{align*}
\]

8. Related work

Due to space limitations, we only describe a few approaches related to our work.

Two approaches are distinguished when description logics are combined with predicate logics: the DL-log approach, where description logics concepts and roles can be used in the body of Datalog programs to infer new Datalog facts (but not new description logic knowledge), and the axiomatic approach (like the one in this paper), where description logics are mapped onto Datalog programs.

⁶Actual classification schemes would be more complex, of course.
The DL-log approach has been introduced by $\mathcal{AL} − \text{Log}^3$, which integrates the $\mathcal{ALC}$ description logics into Datalog programs. Datalog rule bodies can contain concepts which constrain a variable to run over the set of instances of that concept. The rule heads can only contain Datalog predicates, so that it is not able to infer new instances or role fillers. SLD-resolution in combination with an inference method for the description logic part is used for query answering. A more powerful approach is presented by Eiter et.al.$^6$, where the expressive $\mathcal{SHIQ}(D)$ (i.e., OWL Lite) and $\mathcal{SHOIN}(D)$ (i.e., OWL DL) description logics are combined with logic programming under the answer set semantics. In contrast to $\mathcal{AL} − \text{Log}$, this approach does not only provide an information flow from the description logics to the logic program, but also a limited flow back. Straccia$^{28}$ proposes to combine the $\mathcal{ALC}$ description logics with disjunctive logic programs, where (in contrast to other approaches) concepts and roles are also allowed in the rule head. The resulting description logic program language is then extended by uncertainty, following the Generalised Annotated Logic Programming paradigm; thus, probabilities are given as intervals.

Hustadt et.al.$^{14}$ follow the axiomatic approach and transform the $\mathcal{SHIQ}$ description logic into a Disjunctive Datalog$^5$ program, which can be used for query answering. In a first step, the knowledge base is transformed into first-order clauses. Then, basic superposition is applied, and the resulting clauses are converted into Disjunctive Datalog. Alternatively, conceptual Logic Programs have been proposed for converting description logic knowledge bases into logical programs using answer set programming$^{11}$.

Grosof et.al.$^{10}$ proposes a simple mapping a restricted version of $\mathcal{SHOIQ}$ onto Datalog; OWL Lite$^+$ and its mapping onto Datalog has been introduced by de Bruijn et.al.$^2$. In a very similar way, Nottelmann and Fuhr$^{23}$ partially mapped DAML+OIL, the predecessor of OWL, onto four-valued probabilistic Datalog.

P-log$^1$ is a probabilistic extension of answer set programming. P-log provides probabilistic facts and rules, and employs possible world semantics with an independence assumption (as probabilistic Datalog does).

P-$\mathcal{SHOQ}(D)$ is a direct probabilistic extension to description logics$^9$. Similar to our approach, it allows to state the probability that an instance of one concept is also an instance of another concept, that individuals are instances of concepts or related to other individuals by roles; in addition, it is possible to specify that an instance of one concept is related to an individual by a given role. Probabilities are specified as intervals. P-Classic$^{16}$, in contrast, employs Bayesian networks in combination with an independence assumption for inference and computing point probabilities. Jaeger$^{15}$ also allows for probabilistic assertions and terminological knowledge, but employs cross-entropy for defining the semantics, and also derives probability intervals.

Recently, Lukasiewicz$^{20}$ introduced probabilistic description logic programs. These programs can use description logic concepts and roles in rule bodies only, but allow for specifying intervals for conditional probabilities. Choice sets, based on a similar idea as possible worlds, are employed together with an independence assumption for defining the semantics of probabilistic description logic programs.
9. Conclusion and outlook

Popular description logics like OWL Lite are useful for defining ontologies (types, objects and their properties). However, two important features, uncertainty and rules, are missing in standard description logics as well as in OWL Lite for advanced applications like information retrieval. For extending the OWL Lite language in these directions, this paper proposes two new languages, pOWL Lite$^-$ and pOWL Lite$^{EQ}$, which are the first ones capable to deal with uncertain knowledge, and which provide uncertain inference mechanism. We follow the axiomatic approach for combining description logics with Horn predicate logics. Thus, OWL Lite knowledge bases are mapped onto probabilistic Datalog programs. The semantics of the extended languages pOWL Lite$^-$ and pOWL Lite$^{EQ}$ are then defined by the semantics of the resulting probabilistic Datalog model.

OWL Lite is equivalent to the $SHIQ(D)$ description logics, and not all of its features can be mapped onto Datalog. By switching to Datalog$^{EQ}$, however, a larger subset of OWL Lite, namely OWL Lite$^{EQ}$, can be mapped compared to approaches with plain Datalog$^{10}$. The mapping from the whole OWL Lite language onto Datalog variants is sound but not complete (i.e., some inferences are missed). However, integrity constraints provided by Datalog$^{IC,EQ}$ allow for detecting inconsistencies, i.e. each Datalog$^{IC,EQ}$ model is also a description logic model (while it is also possible that the Datalog program is inconsistent although the knowledge is satisfiable). Thus, our mapping is not computing wrong models (which would be misleading), but may miss some inferences.

We also showed that pOWL Lite$^-$ is useful in a practical environment. In an IR example, we are able to cope with uncertain knowledge extracted by wrappers and with uncertain roles for mapping between heterogeneous ontologies, and to classify objects and answer typical IR queries with pOWL Lite.

In the future, we would like to extend the fraction of OWL which can be mapped. Switching to OWL DL, however, poses new problems as Datalog does not allow for disjunction in the head, which is required for modelling the disjunction of concepts. A solution might be switching to disjunctive Datalog, or employing one of the other approaches for combining description logics with logic programming.

We are currently working on a prototype for parsing pOWL Lite$^-$ and pOWL Lite$^{EQ}$ models, mapping them onto probabilistic Datalog, and for evaluating the resulting programs in an IR application.

Acknowledgements

This work is supported by the DFG (grant BIB47 DOuv 02-01, PEPPER).

We wish to thank Umberto Straccia for fruitful discussions. The constructive comments of the reviewers helped improving the paper significantly.

References

20. T. Lukasiewicz.