Uncertainty and Description Logic Programs over Lattices

Umberto Straccia
ISTI-CNR
Via G. Moruzzi 1, I-56124 Pisa ITALY
straccia@isti.cnr.it

Abstract

It is generally accepted that knowledge based systems would be smarter if they can manage uncertainty. In this paper we extend Description Logics, well-known logics for managing structured knowledge, towards the management of uncertainty. We allow (*i*) to express that a sentence is not just true or false, but certain to some degree, which is taken from a certainty lattice; and (*ii*) combine the logic with annotated logic programming, in which the management of uncertainty is based on so-called annotation terms.

Keywords: description logics, ontologies, logic programming, lattices

1 Introduction

In the last decade a substantial amount of work has been carried out in the context of *Description Logics* (DLs) [2]. DLs are a logical reconstruction of the so-called frame-based knowledge representation languages, with the aim of providing a simple well-established Tarski-style declarative semantics to capture the meaning of the most popular features of structured representation of knowledge. Nowadays, a whole family of knowledge representation systems has been build using DLs, which differ with respect to their expressiveness and their complexity, and they have been used for building a variety of applications (see the DL community home page http://dl.kr.org/).

Nowadays, DLs have gained even more popularity due to their application in the context of the *Semantic Web* [28]. *Ontologies* play a key role in the Semantic Web and consists of a hierarchical description of important concepts in a particular domain, along with the description of the properties (of the instances) of each concept. Web content is then annotated by relying on the concepts defined in a specific domain ontology. DLs play a particular role in this context as they are essentially the theoretical counterpart of the *Web Ontology Language OWL DL*, a state of the art language to specify ontologies.

On top of the Ontology layer, the Rules layer of the Semantic Web is going to be developed next (see also [26]), which should offer sophisticated representation and reasoning capabilities. A first effort in this direction is RuleML [5] , fostering an XML-based markup language for rules and rule-based systems, while the OWL Rules Language [27] is a first proposal for extending OWL by Horn clause rules (see Section refrelwork).

However, despite the growing popularity concerning the integration of rule languages, notably Logic Programs [38] (LPs), with DLs (see *e.g.* [12, 27, 37], to the best of our knowledge, no work deals with the combination of DLs with LPs towards the management of uncertainty. This is a well-known and important issue whenever the real world information to be represented is of imperfect nature (unfortunately, this rather the rule than an exception). While both uncertainty in DLs as well as uncertainty in LPs has been addressed, a combination of them in this direction has not yet been considered (for DLs, we may mention for probability theory [20, 22, 30, 34, 54], for possibility theory [25], metric spaces [41], for many-valued [50, 55, 62] and for fuzzy theory [6, 24, 43, 52, 56, 57, 58, 63, 59, 60, 67, 71]; concerning logic programming, we may mention for probability theory [40, 49], for fuzzy set theory [69], multi-valued logic [8, 33, 36, 39, 42], possibilistic logic [13].

The topic of this paper is to combine DLs, LPs and the management of uncertainty into an uniform framework. In particular, we extend DLs towards the management of uncertainty, by allowing to express that a sentence is not just true or false like in classical DLs, but certain to some degree, which is taken from a certainty lattice. The certainty degree dictates to which extend (how certain it is that) a sentence is true. The adopted approach is more general that the fuzzy logic based approach [58], as it subsumes it (just take the lattice over the real unit interval $[0, 1]$ with order $\leq$), but is orthogonal to almost all other approaches. A feature of the lattice approach is that it gives us the possibility to address both *quantitative* uncertainty reasoning (by relying *e.g.* on $[0, 1]$ or subsets of rational numbers like $\{0, \frac{1}{n-1}, \ldots \frac{n-2}{n-1}, 1\}$, for natural number n), as well as *qualitative* uncertainty reasoning (by relying *e.g.* on {`false`, `likelyfalse`, `unknown`, `likelytrue`, `true`}, in increasing order). From a computational point of view, it is still possible to develop a tableaux calculus in the style of almost all DLs and, under reasonable conditions, the computational complexity does not change, which is especially important as usually, reasoning under uncertainty is more involved than the classical case (see, *e.g.* [51]). We then extend the DL with rules, where the management of uncertainty is based on so-called annotation terms, inspired by Generalized Annotated Logic Programming framework of Kifer and Subrahmanian [33], (a quite general approach for managing uncertain information).

We proceed as follows. In the next section, we recall some fundamental notions about DLs and extend them with rules. In Section 3 we describe our DL extension to manage uncertain sentences and address the computational aspect of reasoning in it, while in Section 4 we extend it with annotated logic programming. Finally, Section 5 presents related work, while Section 6 concludes the paper.

2 Preliminaries

2.1 A Quick Look to $\mathcal{ALC}$

The specific DL we extend with uncertainty capabilities is $\mathcal{ALC}$, a significant representative of DLs [2]. Consider three alphabets of symbols, *primitive concepts* (denoted A), *primitive roles* (denoted R) and *individuals* (denoted a and b) [1]. A *concept* (denoted C or D) of the language $\mathcal{ALC}$ is build out from *primitive concepts* A, the *top concept* $\top$, the *bottom concept* $\bot$ and according the following syntax rule:

$$\begin{array}{rcll} C, D & \longrightarrow & C \sqcap D| & \text{(concept conjunction)} \\ & & C \sqcup D| & \text{(concept disjunction)} \\ & & \neg C| & \text{(concept negation)} \\ & & \forall R.C| & \text{(universal quantification)} \\ & & \exists R.C & \text{(existential quantification).} \end{array}$$

An *interpretation* $\mathcal{I}$ is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consisting of a non empty set $\Delta^{\mathcal{I}}$ (called the *domain*) and of an *interpretation function* $\cdot^{\mathcal{I}}$ mapping different individuals into different elements of $\Delta^{\mathcal{I}}$ (called *unique name assumption*), primitive concepts into subsets of $\Delta^{\mathcal{I}}$ and primitive roles into subsets of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation of complex concepts is defined as usual:

$$\begin{array}{lcl} \top^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \\ \bot^{\mathcal{I}} & = & \emptyset \\ (C \sqcap D)^{\mathcal{I}} & = & C^{\mathcal{I}} \cap D^{\mathcal{I}} \\ (C \sqcup D)^{\mathcal{I}} & = & C^{\mathcal{I}} \cup D^{\mathcal{I}} \\ (\neg C)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\ (\forall R.C)^{\mathcal{I}} & = & \{d \in \Delta^{\mathcal{I}} : \forall d'.(d, d') \notin R^{\mathcal{I}} \text{ or } d' \in C^{\mathcal{I}}\} \\ (\exists R.C)^{\mathcal{I}} & = & \{d \in \Delta^{\mathcal{I}} : \exists d'.(d, d') \in R^{\mathcal{I}} \text{ and } d' \in C^{\mathcal{I}}\} \ . \end{array}$$

Two concepts C and D are *equivalent* (denoted $C \equiv D$) when $C^{\mathcal{I}} = D^{\mathcal{I}}$ for all interpretations $\mathcal{I}$ (*e.g.* $\exists R.C \equiv \neg\forall R.\neg C$). An *assertion* (denoted α) is an expression $a{:}C$ ("a is an instance of C"), or an expression $(a, b){:}R$ ("(a, b) is an instance of R"). A *primitive assertion* is either an assertion of the form $a{:}A$, where A is a primitive concept, or an assertion of the form $(a, b){:}R$. An interpretation $\mathcal{I}$ *satisfies* $a{:}C$ (resp. $(a, b){:}R$) iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$ (resp. $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$). Let A and C be a primitive concept and a concept, respectively. A *terminological axiom* (denoted τ) is either a concept specialization or a concept definition. A *concept specialization* is an expression of the form $A{\prec}C$, while a *concept definition* is an expression of the form $A{:=}C$. An interpretation $\mathcal{I}$ *satisfies* $A{\prec}C$ iff $A^{\mathcal{I}} \subseteq C^{\mathcal{I}}$, while $\mathcal{I}$ *satisfies* $A{:=}C$ iff $A^{\mathcal{I}} = C^{\mathcal{I}}$. A finite set K of assertions and terminological axioms is a *Knowledge Base* (KB). With K_A we denote the set of assertions in K, whereas with K_T we denote the set of terminological axioms in K, also called a *terminology*. A KB K is *purely assertional* if $K_T = \emptyset$.

[1] Metavariables may have a subscript or a superscript.

Further, we assume that a terminology K_T is such that no concept A appears more than once on the left hand side of a terminological axiom $\tau \in K_T$ and that no cyclic definitions are present in K_T [2]. An interpretation $\mathcal{I}$ *satisfies* (*is a model of*) a KB K iff $\mathcal{I}$ satisfies each element in K. A KB K *entails* an assertion α (denoted $K \models \alpha$) iff every model of K also satisfies α. The problem of determining whether $K \models \alpha$ is called *entailment problem*, while the problem of determining whether K is satisfiable is called *satisfiability problem*. It is well known (see,*e.g.* [2]) that in $\mathcal{ALC}$: (i) $K \models (a,b){:}R$ iff $(a,b){:}R \in K$; and (ii) $K \models a{:}C$ iff $K \cup \{a{:}\neg C\}$ unsatisfiable. Note that there exists a well known technique based on constraint propagation solving the satisfiability problem [2]. Furthermore, we can restrict our attention to purely assertional KBs, by expanding K to K' and substituting every primitive concept occurring in K, which is defined in K', with its defining term in K'. Informally, the *expansion of a KB* K is as follows [47]: replace each concept specialization $A{\ll}C \in K_T$ with $A{:=}\, C \sqcap A^*$ (A^* is a new primitive concept); then expand the right-hand side of every concept definition by replacing a primitive concept with its definition until there remain only undefined concepts in the second arguments of concept definitions; and finally, replace in K_A all primitive concepts with their definitions. The transformation has the nice property that $K \models \alpha$ iff $K'_A \models \alpha'$, where α' is obtained by replacing every primitive concept occurring in α, which is defined in K'_T, with its defining term in K'_T. This allows us to restrict our attention to purely assertional KBs only (but, the expansion process can be exponential [47]).

2.2 A quick look to description logic programs

For rules, there are mainly three approaches (see, [17, 18], for an overview), the so-called axiom-based approach (*e.g.* [27, 37]) and the DL-log approach (*e.g.*, [12, 14, 15]) and the autoepistemic approach (*e.g.*, [11, 16]). We are not going to discuss in this section these approaches and defer it to Section 5. We just point out that in this paper we follow the DL-log approach, in which rules may not modify the extension of concepts. Essentially, role names and concept names appearing in K_T *may appear in the body of a rule only (except for representing facts)* and are managed as unary and binary predicates, respectively. This because of the underlying assumption that the terminological component *completely* describes the hierarchical structure in the domain, and, therefore, the rules should not allow to make new inferences about that structure.

Syntax. A *term* is either a constant (denoted a, b, c) or a variable (denoted x, y, z). We assume that the description logic component and the rules component share the same alphabet of constants. An *atom* has the form $P(t_1, \ldots, t_n)$,

[2] We say that A *directly uses* primitive concept B in K_T, if there is $\tau \in K_T$ such that A is on the left hand side of τ and B occurs in the right hand side of τ. Let *uses* be the transitive closure of the relation directly uses in K_T. K_T is *cyclic* iff there is A such that A uses A in K_T.

where P is a predicate symbol and t_i are terms. A rule is of the form

$$H \leftarrow B_1, \ldots, B_n \ ,$$

where H and all B_i are atoms. H is called the *rule head*, and the set of all B_i is called the *rule body*. For a rule r, with $H(r)$ and $B(r)$ we indicate the rule head and the rule body or r, respectively. We assume that no rule head atom belongs to the DL signature. For ease the readability, DL predicates will have a DL superscript in the rules. An example of rule is

$$\texttt{MatureFruit}(\texttt{x}) \leftarrow \texttt{Fruit}^{\texttt{DL}}(\texttt{x}), \texttt{Taste}(\texttt{x}, \texttt{sweet})$$

Here `Fruit` is a DL predicate, while `MatureFruit` and `Taste` are not.

Note that [14] allows to use DL axioms in place of DL atoms in the rule body (*e.g.* a concept inclusion may appear in the body of the rule). For easy, we will not deal with this feature.

A *logic program*, denoted $\mathcal{P}$, is a finite set of rules. A *Description Logic Program* (DLP) is a tuple $\mathcal{D}P = \langle K, \mathcal{P} \rangle$, where K is a DL knowledge base and $\mathcal{P}$ is a logic program. For instance, consider the description logic component

$$K = \{\texttt{apple:Fruit}, \texttt{pear:Fruit}, \texttt{a:Fruit} \sqcup \texttt{Cake}\} \ ,$$

and the logic program component

$$\begin{array}{rl} \mathcal{P} \quad = \{ & \texttt{MatureFruit}(\texttt{x}) \leftarrow \texttt{Fruit}^{\texttt{DL}}(\texttt{x}), \texttt{Taste}(\texttt{x}, \texttt{sweet}) \\ & \texttt{Taste}(\texttt{apple}, \texttt{sweet}) \leftarrow \\ & \texttt{Taste}(\texttt{a}, \texttt{sweet}) \leftarrow \\ \} & \ , \end{array}$$

then $\mathcal{D}P = \langle K, \mathcal{P} \rangle$ is a DLP.

Semantics. Essentially, a DL atom appearing in a rule body acts as a query to the underlying DL knowledge base (see [14]). So, consider a DLP $\mathcal{D}P = \langle K, \mathcal{P} \rangle$. The *Herbrand universe* of $\mathcal{P}$, denoted $H_{\mathcal{P}}$ is the set of constants appearing in $\mathcal{D}P$ (if no such constant symbol exists, $H_{\mathcal{P}} = \{c\}$ for an arbitrary constant symbol c from the alphabet of constants). The *Herbrand base* of $\mathcal{P}$, denoted $\mathcal{B}_{\mathcal{P}}$, is the set of all ground atoms built up from the non-DL predicates and the Herbrand universe of $\mathcal{P}$. With $\texttt{ground}(\mathcal{P})$ we denote the grounding of $\mathcal{P}$ w.r.t. $H_{\mathcal{P}}$. For instance, for the DLP above, $H_{\mathcal{P}} = \{\texttt{apple}, \texttt{pear}, \texttt{a}, \texttt{sweet}\}$, $\mathcal{B}_{\mathcal{P}}$ is the set of all ground atoms that can be built from the predicates `MatureFruit` and `Taste` using the constants in $H_{\mathcal{P}}$, while $\texttt{ground}(\mathcal{P})$ is the set of ground instances of rules in $\mathcal{P}$ that can be built using constants in $H_{\mathcal{P}}$. Thus, *e.g.*

$$\begin{array}{l} \texttt{MatureFruit}(\texttt{apple}) \leftarrow \texttt{Fruit}^{\texttt{DL}}(\texttt{apple}), \texttt{Taste}(\texttt{apple}, \texttt{sweet}) \\ \texttt{MatureFruit}(\texttt{a}) \leftarrow \texttt{Fruit}^{\texttt{DL}}(\texttt{a}), \texttt{Taste}(\texttt{a}, \texttt{sweet}) \end{array}$$

belong to $\texttt{ground}(\mathcal{P})$.

An *interpretation* I w.r.t. $\mathcal{D}P$ is a subset of $\mathcal{B}_{\mathcal{P}}$. We say I is a *model* of a ground atom A w.r.t. K, denoted $I \models_K A$, iff

- $A \in I$, if $A \in \mathcal{B}_\mathcal{P}$ (A is a non-DL atom);
- $K \models A$, if A is a DL atom .

We say that I is a *model* of a ground rule r iff $I \models_K H(r)$ whenever $I \models_K B_i$ for all $B_i \in B(r)$. I is *model* of $\mathcal{D}P = \langle K, \mathcal{P} \rangle$ iff $I \models_K r$ for all rules $r \in \texttt{ground}(\mathcal{P})$. We say that a DLP is *satisfiable* if it has a model. Finally, we say that $\mathcal{D}P = \langle K, \mathcal{P} \rangle$ *entails* a ground atom A, denoted $\mathcal{D}P \models A$, iff $I \models_K A$ whenever $I \models \mathcal{D}P$. For instance, given the DLP above,

$$I_1 = \{\texttt{MatureFruit(apple)}, \texttt{Taste(apple, sweet)}, \texttt{Taste(a, sweet)}\}$$

is a model of $\mathcal{D}P$, while

$$I_2 = I_1 \cup \{\texttt{MatureFruit(a)}\}$$

is not. In the latter case note that $I_2 \not\models_K \texttt{Fruit(a)}$ as $K \not\models \texttt{Fruit(a)}$ and, thus, I_2 is not a model of the ground rule

$$\texttt{MatureFruit(a)} \leftarrow \texttt{Fruit}^{\texttt{DL}}\texttt{(a)}, \texttt{Taste(a, sweet)}$$

Finally, note that $\mathcal{D}P \models \texttt{MatureFruit(apple)}$, $\mathcal{D}P \not\models \texttt{MatureFruit(a)}$, $\mathcal{D}P \not\models \texttt{MatureFruit(sweet)}$ and $\mathcal{D}P \not\models \texttt{MatureFruit(pear)}$ hold.

It is easy to see that the entailment problem of a DL literal in a DLP is independent of the presence of a logic program $\mathcal{P}$. This means that the DL knowledge base is unaffected by the rule system, which can be seen as built on top of the DL knowledge base. With that we mean that if $\mathcal{D}P = \langle K, \mathcal{P} \rangle$ the for any ground DL atom A, $\mathcal{D}P \models A$ iff $K \models A$ (no DL atom can appear in the head of a rule).

Interestingly, it is possible to adapt the standard results of Datalog to our case, which say that a satisfiable description logic program $\mathcal{D}P$ has a minimal model $M_{\mathcal{D}P}$ and entailment can be reduced to model checking in this minimal model.

Proposition 1 ([14]) *Let $\mathcal{D}P = \langle K, \mathcal{P} \rangle$ be a DLP. If $\mathcal{D}P$ is satisfiable, then there exists a unique model $M_{\mathcal{D}P} \subseteq \mathcal{B}_\mathcal{P}$ such that $M_{\mathcal{D}P} \subseteq I$ for all models $I \subseteq \mathcal{B}_\mathcal{P}$ of $\mathcal{D}P$. Furthermore, for any ground atom A, $\mathcal{D}P \models A$ iff $M_{\mathcal{D}P} \models_K A$.*

The minimal model can be computed as the least fixed-point of the following monotone operator. Let $\mathcal{D}P = \langle K, \mathcal{P} \rangle$ be a DLP. Define the operator $T_{\mathcal{D}P}$ on interpretations as follows: for every $I \subseteq \mathcal{B}_\mathcal{P}$, let

$$T_{\mathcal{D}P}(I) = \{H(r) \colon r \in \texttt{ground}(\mathcal{P}), I \models_K B_i \text{ for all } B_i \in B(r)\} .$$

It can be shown that $T_{\mathcal{D}P}$ is monotone, *i.e.* $I \subseteq I' \subseteq \mathcal{B}_\mathcal{P}$ implies $T_{\mathcal{D}P}(I) \subseteq T_{\mathcal{D}P}(I')$, and, thus, by the Knaster-Tarski Theorem [66] T_K has a least fixed-point, denoted $lfp(\mathtt{T}_K)$. Moreover, $lfp(\mathtt{T}_K)$ can be computed by finite fixed-point iteration as the signatures and the involved sets in $\mathcal{D}P$ are finite: for every $I \subseteq \mathcal{B}_\mathcal{P}$, we define $T^i_{\mathcal{D}P}(I) = I$, if $i = 0$, and $T^i_{\mathcal{D}P}(I) = T_{\mathcal{D}P}(T^{i-1}_{\mathcal{D}P}(I))$,

if $i > 0$. Then $lfp(\mathtt{T}_K) = \bigcup_{i=0}^{n} T_{\mathcal{D}P}^i(\emptyset) = T_{\mathcal{D}P}^n(\emptyset)$, for some $n \geq 0$. Finally, $lfp(\mathtt{T}_K) = M_{\mathcal{D}P}$ holds.

From an implementation point of view, given $\mathcal{D}P = \langle K, \mathcal{P}\rangle$, we can compute the set of entailed primitive assertions $F = \{\alpha{:}\, K \models \alpha\}$ first and then add them as facts to $\mathcal{P}$. Then, we can rely on standard theorem proving in Datalog.

Example 1 *Consider the following database schema of a company database. Let $\mathcal{D}P = \langle K, \mathcal{P}\rangle$ be such that*

$$\begin{array}{lll} K & = \{ & \mathtt{Employee} {<} \forall \mathtt{OfficeMate}.\mathtt{Employee}, \mathtt{Manager} {<} \mathtt{Employee}\} \\ \mathcal{P} & = \{ & \mathtt{Manager}(\mathtt{jim}), \mathtt{OfficeMate}(\mathtt{jim}, \mathtt{paul}), \\ & & \mathtt{Q}(x) \leftarrow \mathtt{Employee}(x, y)\} \ . \end{array}$$

$\mathtt{Q}(x)$ *is our query. It turns out that* $\mathcal{D}P \models \mathtt{Q}(\mathtt{jim})$ *and* $\mathcal{D}P \models \mathtt{Q}(\mathtt{paul})$, i.e. `jim` *and* `paul` *are an answer to our query.* □

3 The logic $\mathcal{L}$-$\mathcal{ALC}$

Let $\mathcal{L} = \langle \mathcal{T}, \preceq\rangle$ be a *certainty lattice* (a complete lattice), where $\mathcal{T}$ is a set of certainty values and $\preceq$ is a partial order over $\mathcal{T}$. Let $\otimes$ and $\oplus$ be the meet and join operators induced by $\preceq$, respectively. Let f and t be the least and greatest element in $\mathcal{T}$, respectively. We also assume that there is a function from $\mathcal{T}$ to $\mathcal{T}$, called *negation function* (denoted $\neg$) that is anti-monotone w.r.t. $\preceq$ and satisfies $\neg\neg\alpha = \alpha, \forall\alpha \in \mathcal{T}$. The main idea is that an assertion $a{:}C$, rather being interpreted as either true or false, will be mapped into a certainty value c in $\mathcal{T}$. The intended meaning is that c indicates to which extend (how certain it is that) 'a is a C'. Typical certainty lattice are: given a set of real values $\mathcal{T}$, consider $\mathcal{L}_{\mathcal{T}} = \langle \mathcal{T}, \leq\rangle$. Then $\mathcal{L}_{\{0,1\}}$ corresponds to the classical truth-space, where 0 stands for 'false', while 1 stands for 'true', while $\mathcal{L}_{[0,1]}$, which relies on the unit real interval, is quite frequently used as certainty lattice. In $\mathcal{L}_{[0,1]}$, $\neg\alpha = 1 - \alpha$ is quite typical. Another frequent certainty lattice is Belnap's $\mathcal{FOUR}$ [4], where $\mathcal{T}$ is $\{f, t, u, i\}$ with $f \preceq u \preceq t$ and $f \preceq i \preceq t$. Here, u stands for 'unknown', whereas i stands for inconsistency. We denote the lattice as $\mathcal{L}_B$. Additionally, besides $\neg f = t$, we have $\neg u = u$ and $\neg i = i$. Another certainty lattice is $\mathcal{L}_4$, where $\mathcal{T}$ is $\{f, lf, lt, t\}$ with $f \preceq lf \preceq lt \preceq t$. Here, lf stands for 'likely false', whereas lt stands for 'likely true'. Besides $\neg f = t$, we have $\neg lf = lt$. A further popular lattice allows us to reason about *belief and doubt*. Indeed, the idea is to take any lattice $\mathcal{L}$, and to consider the cartesian product $\mathcal{L} \times \mathcal{L}$. For any pair $(b, d) \in \mathcal{L} \times \mathcal{L}$, b indicates the degree of *belief* a reasoning agent has about a sentence s, while d indicates the degree of *doubt* the agent has about s. The order on $\mathcal{L} \times \mathcal{L}$ is determined by $(b, d) \preceq (b', d')$ iff $b \preceq b'$ and $d' \preceq d$, *i.e.* belief goes up, while doubt goes down. The minimal element is (f, t) (no belief, maximal doubt), while the maximal element is (t, f) (maximal belief, no doubt). Negation is given by $\neg(b, d) = (d, b)$ (exchange belief with doubt). We indicate this lattice with $\bar{\mathcal{L}}$. The examples above illustrates that the

lattice approach gives us the possibility to address both quantitative uncertainty reasoning as well as qualitative uncertainty reasoning.

For a certainty lattice $\mathcal{L} = \langle \mathcal{T}, \preceq \rangle$, an $\mathcal{L}$-*interpretation* is now a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is, as for the classical case, the *domain*, whereas $\cdot^{\mathcal{I}}$ is an *interpretation function* mapping (i) individuals as for the classical case, *i.e.* $a^{\mathcal{I}} \neq b^{\mathcal{I}}$, if $a \neq b$; (ii) a concept C into a function $C^{\mathcal{I}}\colon \Delta^{\mathcal{I}} \rightarrow \mathcal{T}$; (iii) a role R into a function $R^{\mathcal{I}}\colon \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \rightarrow \mathcal{T}$. In the following, for ease with interpretation we mean always an $\mathcal{L}$-interpretation, for some certainty lattice $\mathcal{L}$. As anticipated above, if $d \in \Delta^{\mathcal{I}}$ is an object of the domain $\Delta^{\mathcal{I}}$ then $C^{\mathcal{I}}(d)$ gives us the degree of certainty of being the object d an instance of the concept C under the interpretation $\mathcal{I}$. Similarly for roles. The interpretation function $\cdot^{\mathcal{I}}$ has to satisfy the following equations: for all $d \in \Delta^{\mathcal{I}}$,

$$
\begin{array}{lcl}
\top^{\mathcal{I}}(d) & = & t \\
\bot^{\mathcal{I}}(d) & = & f \\
(C \sqcap D)^{\mathcal{I}}(d) & = & C^{\mathcal{I}}(d) \otimes D^{\mathcal{I}}(d) \\
(C \sqcup D)^{\mathcal{I}}(d) & = & C^{\mathcal{I}}(d) \oplus D^{\mathcal{I}}(d) \\
(\neg C)^{\mathcal{I}}(d) & = & \neg C^{\mathcal{I}}(d) \\
(\forall R.C)^{\mathcal{I}}(d) & = & \bigotimes_{d' \in \Delta^{\mathcal{I}}} \{\neg R^{\mathcal{I}}(d, d') \oplus C^{\mathcal{I}}(d')\} \\
(\exists R.C)^{\mathcal{I}}(d) & = & \bigoplus_{d' \in \Delta^{\mathcal{I}}} \{R^{\mathcal{I}}(d, d') \otimes C^{\mathcal{I}}(d')\}.
\end{array}
$$

Note that the semantics of $\exists R.C$ is the result of viewing $\exists R.C$ as the open first order formula $\exists y.R(x, y) \wedge \bar{C}(y)$ (where $\bar{C}$ is the translation of C into first-order logic) and $\exists$ is viewed as a disjunction over the elements of the domain. Similarly, the semantics of $\forall R.C$ is related to $\forall y.\neg R(x, y) \vee \bar{C}(y)$, where $\forall$ is viewed as a conjunction over the elements of the domain. The definition of concept *equivalence* is like for $\mathcal{ALC}$. As for classical $\mathcal{ALC}$, dual relationships between concepts hold: *e.g.* $\top \equiv \neg \bot$, $(C \sqcap D) \equiv \neg(\neg C \sqcup \neg D)$ and $(\forall R.C) \equiv \neg(\exists R.\neg C)$.

An $\mathcal{L}$-*assertion* (denoted ψ) is an expression $\langle \alpha, c \rangle$, where α is an $\mathcal{ALC}$ assertion and $c \in \mathcal{T}$. From a semantics point of view, an $\mathcal{L}$-assertion $\langle \alpha, c \rangle$ constrains the certainty-value of α to be greater or equal to c. An interpretation $\mathcal{I}$ *satisfies* $\langle a{:}C, c \rangle$ (resp. $\langle (a, b){:}R, c \rangle$) iff $C^{\mathcal{I}}(a^{\mathcal{I}}) \succeq c$ (resp. $R^{\mathcal{I}}(a^{\mathcal{I}}, b^{\mathcal{I}}) \succeq c$). Two $\mathcal{L}$-assertions ψ_1 and ψ_2 are *equivalent* (denoted $\psi_1 \equiv \psi_2$) iff they are satisfied by the same set of interpretations. A *primitive* $\mathcal{L}$-*assertion* is a $\mathcal{L}$-assertion involving a primitive assertion only.

Concerning terminological axioms, an interpretation $\mathcal{I}$ *satisfies* $A{\prec}C$ iff $\forall d \in \Delta^{\mathcal{I}}, A^{\mathcal{I}}(d) \preceq C^{\mathcal{I}}(d)$, while $\mathcal{I}$ *satisfies* $A{:=}C$ iff $\forall d \in \Delta^{\mathcal{I}}, A^{\mathcal{I}}(d) = C^{\mathcal{I}}(d)$. In $\mathcal{L}$-$\mathcal{ALC}$, a *knowledge base* is a finite set of $\mathcal{L}$-assertions and terminological axioms. With K_A we denote the set of $\mathcal{L}$-assertions in K, with K_T we denote the set of terminological axioms in K (the terminology), if $K_T = \emptyset$ then K is *purely assertional*, and we assume that a terminology K_T is such that no concept A appears more than once on the left hand side of a terminological axiom in K_T and that no cyclic definitions are present in K_T. An interpretation $\mathcal{I}$ *satisfies* (is *a model of*) a knowledge base K iff $\mathcal{I}$ satisfies each element of K. A KB K $\mathcal{L}$-*entails* an $\mathcal{L}$-assertion ψ (denoted $K \models_{\mathcal{L}} \psi$) iff every model of K also satisfies

ψ. Finally, given a KB K and an assertion α, it is of interest to compute α's best lower certainty-value bound. To this, the *greatest lower bound* of α w.r.t. K (denoted $glb(K, \alpha)$) is $\bigoplus\{c : K \models_{\mathcal{L}} \langle \alpha, c\rangle\}$ ($\bigoplus \emptyset = f$). Determining the glb is called the *Best Certainty-Value Bound* (BCVB) problem.

In the following we show an application in the context of logic-based information retrieval.

Logic-based multimedia information retrieval. Let us first roughly present (parts of) the LMIR model of [43, 57]. In doing this, we rely on Figure 1 and consider $\mathcal{L}_{[0,1]}$. The model has two layers addressing the multidimensional aspect of multimedia objects $o \in \mathbb{O}$ (*e.g.* objects `o1` and `o2` in Figure 1): that is, their *form* and their *semantics* (or *meaning*). The form of a multimedia object is a collective name for all its *media dependent*, typically automatically extracted features, like text index term weights (object of type text), colour distribution, shape, texture, spatial relationships (object of type image), mosaiced video-frame sequences and time relationships (object of type video). On the other hand, the semantics (or meaning) of a multimedia object is a collective name for those features that pertain to the slice of the real world being *represented*, which exists independently of the existence of a object referring to it. Unlike form, the semantics of a multimedia object is thus *media independent* (typically, constructed manually perhaps with the assistance of some automatic tool).

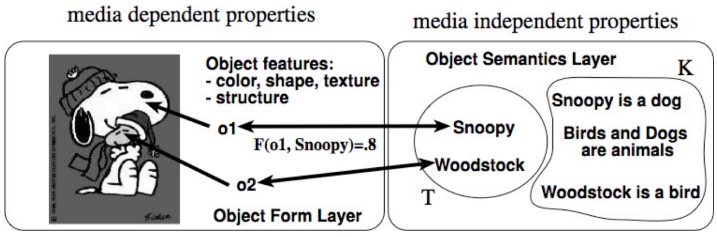

Figure 1: LMIR model layers and objects

Therefore, we have two layers, the *object form layer* and the *object semantics layer*. The former represents media dependent features of the objects, while the latter describes the semantic properties of the slice of world the objects are about. The semantic entities (*e.g.*, Snoopy, Woodstock), which objects can be about are called *semantic index terms* ($t \in \mathbb{T}$). The mapping of objects $o \in \mathbb{O}$ to semantic entities $t \in \mathbb{T}$ (*e.g.*, "object `o1` is about `Snoopy`") is called *semantic annotation*. According to the fuzzy information retrieval model [7, 31, 35, 48], semantic annotation can be formalized as a membership function $F: \mathbb{O} \times \mathbb{T} \rightarrow [0, 1]$ describing the *correlation* between multimedia objects and semantic index terms. The value $F(o, t)$ indicates to which degree the multimedia object o deals with the semantic index term t. The meaning of a semantic index term t may in this context be represented as a fuzzy subset of multimedia objects in $\mathbb{O}$, $m(t)$, with the quantitative measure of aboutness being the values of

function F for a given semantic index term t, *i.e.* $m(t) = \{\langle o, f_t(o)\rangle : o \in \mathbb{O}\}$, in which $f_t(o) = F(o,t)$. $m(t)$ is the meaning of term t. The function F acts as the membership function of $m(t)$. Depending on the context, the function F may be computed automatically (*e.g.*, for text we may have [10], for images we may have an automated image annotation (classification) tool, as *e.g.* [19]). Note that the function F will be a source for assertions of the form $\langle(\mathtt{o},\mathtt{t}){:}\mathtt{F}, F(o,t)\rangle$ (see [43]). In practice, the scenario depicted in Figure 1 may roughly be represented with the following knowledge base K:

$$\begin{array}{rcl} K & \supseteq & \{\mathtt{Bird}{<:}\mathtt{Animal}, \mathtt{Dog}{<:}\mathtt{Animal}\} \cup \\ & & \{\langle \mathtt{snoopy}{:}\mathtt{Dog}, 1\rangle, \langle \mathtt{woodstock}{:}\mathtt{Bird}, 1\rangle, \\ & & \langle(\mathtt{o1}, \mathtt{snoopy}){:}\mathtt{F}, 0.8\rangle, \langle(\mathtt{o2}, \mathtt{woodstock}){:}\mathtt{F}, 0.7\rangle\} \ . \end{array}$$

Now, consider the query concept $Q = \mathtt{Object} \sqcap \exists \mathtt{F}.\mathtt{Animal}$, *i.e.* retrieve all objects about animals. Then it is easily verified that we retrieve both objects, but with different *Retrieval Status Value*s [3], which indicate their relatedness to the query ($K \models \langle \mathtt{snoopy}{:}Q, 0.8\rangle$, $K \models \langle \mathtt{woodstock}{:}Q, 0.7\rangle$).

3.1 Decision algorithms in $\mathcal{L}$-$\mathcal{ALC}$

Deciding satisfiability of a KB requires a calculus. Without loss of generality we consider purely assertional KBs only. We develop a calculus in the style of the constraint propagation method, as this method is usually proposed in the context of DLs [2]. Essentially, it generalizes the calculus presented in [58] for $\mathcal{L}_{[0,1]}$, to any certainty lattice $\mathcal{L}$. We first address the entailment problem and then the BCVB problem.

To guarantee soundness and completeness of the calculus, we make the following restrictions. We assume that in the certainty lattice $\mathcal{L} = \langle \mathcal{T}, \preceq\rangle$ the set of certainty values $\mathcal{T}$ is *finite*. From a practical point of view this is a limitation we can live with, especially taking into account that computers have finite resources, and thus, only a finite set of certainty values can be represented. In particular, this includes also the case of the the rational numbers in $[0,1] \cap \mathbb{Q}$ under a given fixed precision p a computer can work with. We also point out an error in our early version [62], in which we do not rely on this assumption. Without this assumption, the results in [62] do not hold.

At first, we generalize $\mathcal{L}$-assertions to the form $\langle \alpha \succeq c\rangle$ and $\langle \alpha \preceq c\rangle$ with obvious semantics. Of course, $\langle \alpha, c\rangle$ is a syntactic variant of $\langle \alpha \succeq c\rangle$. We also assume that an assertion is always in *Negation Normal Form* where a negation appears in the head of an atom only. Furthermore, an expression $\langle a{:}\neg A \succeq c\rangle$ is always replaced with the expression $\langle a{:}A \preceq \neg c\rangle$. Similarly, an expression $\langle a{:}\neg A \preceq c\rangle$ is replaced with the expression $\langle a{:}A \succeq \neg c\rangle$.

An $\mathcal{L}$-*constraint* (denoted ψ) is inductively defined as follows: (*i*) an $\mathcal{L}$-assertion is an $\mathcal{L}$-constraint; (*ii*) if ψ and ψ' are $\mathcal{L}$-constraints, then so are $\neg\psi, \psi \wedge \psi'$ and $\psi \vee \psi'$ (*e.g.* $\langle \alpha_1 \preceq c_1\rangle \vee \neg\langle \alpha_2 \succeq c_2\rangle$). A *literal* is a primitive $\mathcal{L}$-assertion or its boolean negation. Without loss of generality, we assume that $\mathcal{L}$-constraints are always in *Negation Norma Form* (NNF), where a boolean negation appears

Table 1: Conjugated pairs.

	$\langle \alpha \not\succeq c' \rangle$	$\langle \alpha \preceq c' \rangle$
$\langle \alpha \succeq c \rangle$	$\neg(\exists c''.c'' \succeq c \wedge c'' \not\succeq c')$	$c \not\preceq c'$
$\langle \alpha \not\preceq c \rangle$	$\neg(\exists c''.c'' \not\preceq c \wedge c'' \not\succeq c')$	$\neg(\exists c''.c'' \not\preceq c \wedge c'' \preceq c')$

in the head of an $\mathcal{L}$-assertion only. The definition of satisfiability (of a set) of $\mathcal{L}$-constraints is easy: *e.g.* I satisfies $\neg\psi$ iff I does not satisfy ψ, while I satisfies $\psi \vee \psi'$ iff I does satisfy either ψ or ψ'. It follows that

$$K \models_{\mathcal{L}} \langle \alpha \succeq c \rangle \text{ iff } K \cup \{\neg\langle \alpha \succeq c \rangle\} \text{ not satisfiable.}$$

Therefore, it is sufficient to develop a calculus for the satisfiability problem.

Note that in case $\mathcal{L}$ is a total order, then we have the equivalences between $\neg\langle \alpha \succeq c \rangle$ and $\langle \alpha \prec c \rangle$, and between $\neg\langle \alpha \preceq c \rangle$ and $\langle \alpha \succ c \rangle$. For instance, this property is used in the calculus developed for fuzzy $\mathcal{ALC}$ [58]. In general, these equivalences do not hold (*e.g.*, in $\mathcal{L}_B$, $\neg\langle \alpha \succeq u \rangle$ is equivalent to $\langle \alpha \preceq i \rangle$ and not to $\langle \alpha \prec u \rangle$). For ease, sometimes we write $\langle \alpha \not\succeq c \rangle$ in place of $\neg\langle \alpha \succeq c \rangle$ and $\langle \alpha \not\preceq c \rangle$ in place of $\neg\langle \alpha \preceq c \rangle$.

Our calculus, determining whether a finite set S of $\mathcal{L}$-constraints is satisfiable or not, is based on a set of constraint propagation rules transforming a set S into "simpler" satisfiability preserving sets S_i until either all S_i contain an inconsistency, called *clash* (indicating that from all the S_i no model of S can be build), or some S_i is completed and clash-free, that is, no rule can be further be applied to S_i and S_i contains no clash (indicating that from S_i a model of S can be build). A set of $\mathcal{L}$-constraints S contains a *clash* (inconsistency) iff it contains a set of literals $\{\langle \alpha\ r_j\ c_j \rangle\}_{j \in J}$ (with $r_j \in \{\succeq, \preceq, \not\succeq, \not\preceq\}$) such that the set of constraints $\{\langle x\ r_j\ c_j \rangle\}_{j \in J}$ has no solution for the variable x in the lattice $\mathcal{L}$[3]. For instance, S contains a clash if it contains either $\langle a{:}\bot \succeq c \rangle$ (where $c \succ f$), or $\langle a{:}\top \preceq c \rangle$ (where $c \prec t$), or $\langle a{:}\bot \not\preceq c \rangle$, or $\langle a{:}\top \not\succeq c \rangle$, or $\langle \alpha \not\succeq f \rangle$, or $\langle \alpha \not\preceq t \rangle$, or S contains a conjugated pair of $\mathcal{L}$-constraints. Each entry in the Table 1 says us under which condition the row-column pair of $\mathcal{L}$-constraints is a *conjugated pair*. For instance, in $\mathcal{L}_B$, $\langle a{:}A \succeq i \rangle$ and $\langle a{:}A \preceq u \rangle$ is a conjugated pair as $i \not\preceq u$. While in total ordered lattices checking inconsistency is easy, this may not be the case for arbitrary lattices. Given an $\mathcal{L}$-constraint ψ, with ψ^c we indicate a conjugate of ψ (if there exists one). A conjugate of an $\mathcal{L}$-constraint may be not unique, as there could be infinitely many. For instance, in $\mathcal{L}_{[0,1]}$-$\mathcal{ALC}$, any $\langle a{:}A \preceq c \rangle$ with $c \prec c'$ is a conjugate of $\langle a{:}A \succeq c' \rangle$.

There are obvious tableaux rules for the boolean connectives $\wedge$ and $\vee$. For each connective $\sqcap, \sqcup, \neg, \forall, \exists$ there is a rule for each relation $\succeq, \preceq, \not\succeq$ and $\not\preceq$. In what follows, for any $c \in \mathcal{T}$, with $D_{\mathcal{L}}(c)$ we indicate the set $D_{\mathcal{L}}(c)$ as the set of all minimal elements of the set $\{(c_1, c_2) \in \mathcal{T} \times \mathcal{T} : c_1 \oplus c_2 \succeq c\}$ where the order

[3]A solution for x is a certainty value c such that all constraints in $\{\langle c\ r_j\ c_j \rangle\}_{j \in J}$ are satisfied, *i.e.* for all $j \in J$, $c\ r_j\ c_j$ holds in $\mathcal{L}$.

over pairs is $(c_1, c_2) \preceq (c_3, c_4)$ iff $c_1 \preceq c_3$ and $c_2 \preceq c_4$. The purpose of $D_{\mathcal{L}}(c)$ is to identify meaningful candidates c_1 and c_2, making *e.g.* a disjunction of the form $\langle a{:}A \sqcup B \succeq c\rangle$ true, whenever we reduce $\langle a{:}A \sqcup B \succeq c\rangle$ to $\{\langle a{:}A \succeq c_1\rangle \wedge \langle a{:}B \succeq c_2\rangle\}$. The choice is non-deterministic and there could $|D_{\mathcal{L}}(c)|$ many choices, and $|D_{\mathcal{L}}(c)|$ depends on c and $\mathcal{L}$. Note that $(c_1, c_2) \in D_{\mathcal{L}}(c)$ iff $(c_2, c_1) \in D_{\mathcal{L}}(c)$ and $|D_{\mathcal{L}}(c)| \succeq 2$ (except the case where $c = f$, in which case $|D_{\mathcal{L}}(c)| = 1$) as $\{(c, f), (f, c)\} \subseteq D_{\mathcal{L}}(c)$, *e.g.* in $\mathcal{L}_{\{0,1\}}$, $D_{\mathcal{L}_{\{0,1\}}}(1) = \{(1, 0), (0, 1)\}$ indicating that *e.g.* $\langle a{:}A \sqcup B \succeq c\rangle$ can be branched into either $\{\langle a{:}A \succeq 1\rangle\}$ or $\{\langle a{:}B \succeq 1\rangle\}$. Note also that in $\mathcal{L}_B$, we have $D_{\mathcal{L}_B}(t) = \{(t, f), (f, t), (u, i), (i, u)\}$ and, thus, *e.g.* $\langle a{:}A \sqcup B \succeq c\rangle$ can be branched into four alternatives. As $\mathcal{L}$ has finite $\mathcal{T}$, then for any $c \in \mathcal{T}$, $D_{\mathcal{L}}(c)$ is finite as well. If $\mathcal{L}$ is a total order, like $\mathcal{L}_4$ or $\mathcal{L}_{[0,1]}$, then $D_{\mathcal{L}}(c) = \{(c, f), (f, c)\}$, for any $c \in \mathcal{T}$.

The limitation to the finiteness of $\mathcal{L}$ will guarantee both the termination of our decision procedure and its correctness and completeness.

We present rules for $\sqcap, \sqcup, \neg, \forall, \exists$, and $\succeq$ and $\not\succeq$ only. The rules for $\preceq, \not\preceq$ can be derived from $\succeq$ and $\not\succeq$, respectively. Indeed, the rules for $\langle a{:}C \preceq c\rangle$, $\langle a{:}C \sqcup D \preceq c\rangle$, $\langle a{:}C \sqcap D \preceq c\rangle$, $\langle a{:}\exists R.C \preceq c\rangle$ and $\langle a{:}\forall R.C \preceq c\rangle$ can be derived from the equivalent expressions $\langle a{:}\neg C \succeq \neg c\rangle$, $\langle a{:}\neg C \sqcap \neg D \succeq \neg c\rangle$, $\langle a{:}\neg C \sqcup \neg D \succeq \neg c\rangle$, $\langle a{:}\forall R.\neg C \succeq \neg c\rangle$ and $\langle a{:}\exists R.\neg C \succeq \neg c\rangle$, respectively. Similarly for $\not\preceq$ (*e.g.* we have the following equivalence: $\langle a{:}C \not\preceq c\rangle \equiv \neg\langle a{:}C \preceq c\rangle \equiv \neg\langle a{:}\neg C \succeq \neg c\rangle \equiv \langle a{:}\neg C \not\succeq \neg c\rangle$).

The rules below rely on the following properties over a lattice $\mathcal{L} = \langle \mathcal{T}, \preceq\rangle$: for any $c, c_1, c_2 \in \mathcal{T}$, (i) $c_1 \otimes c_2 \succeq c$ iff $c_1 \succeq c$ and $c_2 \succeq c$; (ii) $c_1 \otimes c_2 \not\succeq c$ iff $c_1 \not\succeq c$ or $c_2 \not\succeq c$; (iii) $c_1 \oplus c_2 \succeq c$ iff $c_1 \succeq c'$ and $c_2 \succeq c''$, for $(c', c'') \in D_{\mathcal{L}}(c)$; and (iv) $c_1 \oplus c_2 \not\succeq c$ iff not $(c_1 \oplus c_2 \succeq c)$ iff for all $(c', c'') \in D_{\mathcal{L}}(c)$, either $c_1 \not\succeq c'$ or $c_2 \not\succeq c''$.

In the following rules we assume that a new constraint is added to a constraint set S if it is not subsumed in S (a constraint $\langle \alpha \succeq c\rangle$ is *subsumed* by a constraint $\langle \alpha \succeq c'\rangle$ iff $c' \succeq c$ -similarly for the other relations $\not\succeq, \preceq$, and $\not\preceq$). We also avoid adding constraints of the form $\langle \alpha \succeq f\rangle$ and $\langle \alpha \preceq t\rangle$.

1. $S \rightarrow_{\sqcap_\succeq} S \cup \{\langle a{:}C \succeq c\rangle \wedge \langle a{:}D \succeq c\rangle\}$
   if $\psi = \langle a{:}C \sqcap D \succeq c\rangle \in S$ and the $(\sqcap_\succeq)$ rule has not yet been applied to $\psi \in S$

2. $S \rightarrow_{\sqcap_{\not\succeq}} S \cup \{\langle a{:}C \not\succeq c\rangle \vee \langle a{:}D \not\succeq c\rangle\}$
   if $\psi = \langle a{:}C \sqcap D \not\succeq c\rangle \in S$ and the $(\sqcap_{\not\succeq})$ rule has not yet been applied to $\psi \in S$

3. $S \rightarrow_{\sqcup_\succeq} S \cup \{\psi'\}$
   if $\psi = \langle a{:}C_1 \sqcup C_2 \succeq c\rangle \in S$, the $(\sqcup_\succeq)$ rule has not yet been applied to $\psi \in S$ and $\psi' = \bigvee_{(c_1,c_2)\in D_{\mathcal{L}}(c)} \langle a{:}C_1 \succeq c_1\rangle \wedge \langle a{:}C_2 \succeq c_2\rangle$

4. $S \rightarrow_{\sqcup_{\not\succeq}} S \cup \{\psi'\}$
   if $\psi = \langle a{:}C_1 \sqcup C_2 \not\succeq c\rangle \in S$, the $(\sqcup_{\not\succeq})$ rule has not yet been applied to $\psi \in S$ and $\psi' = \bigwedge_{(c_1,c_2)\in D_{\mathcal{L}}(c)} (\langle a{:}C_1 \not\succeq c_1\rangle \vee \langle a{:}C_2 \not\succeq c_2\rangle)$

5. $S \rightarrow_{\forall_{\succeq}} S \cup \{\psi'\}$
   if $\{\langle a{:}\forall R.C \succeq c\rangle, \psi^c\} \subseteq S$, $\psi = \langle (a,b){:}R \preceq \neg c\rangle$, the $(\forall_{\succeq})$ rule has not yet been applied in S to the pair $(\langle a{:}\forall R.C \succeq c\rangle, \psi)$ and ψ' is the expression $\bigvee_{(c_1,c_2)\in D_{\mathcal{L}}(c)\setminus\{(c,f)\}} \langle (a,b){:}R \preceq \neg c_1\rangle \wedge \langle b{:}C \succeq c_2\rangle$

6. $S \rightarrow_{\forall_{\not\succeq}} S \cup \{\psi'\}$
   if $\psi = \langle a{:}\forall R.C \not\succeq c\rangle, \in S$, the $(\forall_{\not\succeq})$ rule has not yet been applied to $\psi \in S$ and $\psi' = \bigwedge_{(c_1,c_2)\in D_{\mathcal{L}}(c)} \langle (a,b){:}R \not\preceq \neg c_1\rangle \vee \langle b{:}C \not\succeq c_2\rangle$, for a new constant b

7. $S \rightarrow_{\exists_{\succeq}} S \cup \{\langle (a,b){:}R \succeq c\rangle \wedge \langle b{:}C \succeq c\rangle\}$
   if $\psi = \langle a{:}\exists R.C \succeq c\rangle \in S$, b new constant and the $(\exists_{\succeq})$ rule has not yet been applied to $\psi \in S$

8. $S \rightarrow_{\exists_{\not\succeq}} S \cup \{\psi'\}$
   if $\{\langle a{:}\exists R.C \succeq c\rangle, \psi^c\} \subseteq S$, $\psi = \langle (a,b){:}R \not\succeq c\rangle$, the $(\exists_{\not\succeq})$ rule has not yet been applied in S to the pair $(\langle a{:}\exists R.C \succeq c\rangle, \psi)$ and $\psi' = \langle b{:}C \not\succeq c_2\rangle$).

Some of the above rules deserve some explanation. The $(\sqcup_{\succeq})$ rule is a generalization of the classical disjunction rule. The main difference is that, depending on the lattice $\mathcal{L}$, there may be more than the usual two branches (likely as many as $|D_{\mathcal{L}}(c)|$). For instance, in $\mathcal{L}_B$, a constraint $\langle a{:}C_1 \sqcup C_2 \succeq t\rangle \in S$ may give rise to four branches, each containing $\langle a{:}C_1 \succeq t\rangle$, $\langle a{:}C_2 \succeq t\rangle$, $\langle a{:}C_1 \succeq i\rangle \wedge \langle a{:}C_2 \succeq u\rangle$ and $\langle a{:}C_1 \succeq u\rangle \wedge \langle w{:}C_2 \succeq i\rangle$, respectively. Note that if $\mathcal{L}$ is a total order then there are at most two branches. The $(\sqcup_{\not\succeq})$ rule is derived from the $(\sqcup_{\succeq})$ rule. The $\forall_{\succeq}$ is a specialization of the $(\sqcup_{\succeq})$ rule as well and is a generalization of the classical rule for $\forall$. Indeed, according to the semantics of $\mathcal{L}$-$\mathcal{ALC}$, $\langle a{:}\forall R.C \succeq c\rangle$ may be viewed as a disjunction and with decomposition rule

- $S \rightarrow_{\forall'_{\succeq}} S \cup \{\langle (a,b){:}R \preceq \neg c_1\rangle, \langle b{:}C \succeq c_2\rangle\}$
  if $\langle a{:}\forall R.C \succeq c\rangle \in S$, b occurs in S and $(c_1,c_2) \in D_{\mathcal{L}}(c)$

But, observing that $(c,f) \in D_{\mathcal{L}}(c)$, we obtain a constraint set S containing $\psi = \langle (a,b){:}R \preceq \neg c\rangle$ (*e.g.*, in the classical $\mathcal{L}_{\{t,f\}}$, with $c = t$, we have $\langle (a,b){:}R \preceq f\rangle \in S$). This constraint can only be clashed if S contains a conjugate to ψ (*e.g.*, in the classical case $\langle (a,b){:}R \succeq t\rangle$ must be in S). This motivates the $\forall_{\succeq}$ rule as a refinement of the above $\forall'_{\succeq}$ rule. Note that the $(\forall_{\not\succeq})$ rule can be worked out by a similar argumentation, by relying on the $(\sqcup_{\not\succeq})$ rule, and is left to the reader.

A constraint set S is *complete* if no rule is applicable to it. A complete set S_2 obtained from a set S_1 by applying the above rules is called a *completion* of S_1. Note that more than one completion can be obtained. It can be verified that for finite certainty lattices, the above calculus has the *termination property*, *i.e.* any completion of a finite set of $\mathcal{L}$-constraints S can be obtained after a finite number of rule applications.

Moreover, in a similar way as in [58], Proposition 3, it is easily verified, by case analysis, that the above rules are sound, *i.e.* if S_1 is satisfiable then there is a satisfiable completion S_2 of S_1. Here we use the fact that for finite lattices, if *e.g.*

$$(\exists R.C)^{\mathcal{I}}(d) = \bigoplus_{d'\in\Delta^{\mathcal{I}}} \{R^{\mathcal{I}}(d,d') \otimes C^{\mathcal{I}}(d')\} \succeq c$$

then there is $d' \in \Delta^{\mathcal{I}}$ such that $R^{\mathcal{I}}(d, d') \succeq c$ and $C^{\mathcal{I}}(d') \succeq c$, which motivates rule 7. The case for $\forall R.C$ is similar.

Proposition 2 *For a finite certainty lattice $\mathcal{L}$, a finite set of $\mathcal{L}$-constraints S is satisfiable iff there exists a clash free completion of S.* $\dashv$

From a computational complexity point of view, the satisfiability problem is in the same complexity class (PSPACE-complete [53]) as for $\mathcal{ALC}$. Indeed, let us indicate with $|\mathcal{L}|$ the finite dimension of representing $\mathcal{L}$ and with $|S|$ the dimension of a constraint set S. With *combined complexity* we intend the complexity w.r.t. $|\mathcal{L}| + |S|$.

In a similar way as in [58], Proposition 4:

Proposition 3 *The satisfiability problem is PSPACE-complete w.r.t. combined complexity for finite certainty lattices.* $\dashv$

The above result says us that no additional computational cost has to be paid for the major expressive power (if $\mathcal{L}$ is finite, of course).

Concerning the BCVB problem, for any finite certainty lattice $\mathcal{L} = \langle \mathcal{T}, \preceq \rangle$ we may apply a simple iterative approximation algorithm for determining the *glb*:

1. let $\mathcal{T}_0 = \mathcal{T}$ and $\bar{c} := f$. For $j = 0, 1, 2, \ldots$
2. take a value $c \in \mathcal{T}_j$; if there is no such value then set $glb(K, \alpha) := \bar{c}$ and exit;
3. if $K \models_{\mathcal{L}} \langle \alpha \succeq c \rangle$ then set $\bar{c} := c$ and $\mathcal{T}_{j+1} = \mathcal{T}_j \setminus (\{c' : c' \not\succeq \bar{c}\} \cup \{c\})$, otherwise $\mathcal{T}_{j+1} = \mathcal{T}_j \setminus \{c' : c' \succeq c\}$;
4. go to step 2.

The procedure converges after at most $|\mathcal{T}|$ steps. Of course, the speed of convergence depends on the 'goodness' of the chosen value c in step 2.

4 Uncertainty in description logic programs

We are going now to define an extension of DLPs towards the management of uncertainty. Classically, n-ary predicates may be seen as functions from their domain into $\{0, 1\}$, where 0 stands for $false$, while 1 stands for $true$. We extend this notion by mapping n-ary predicates into functions from their domain into a certainty lattice $\mathcal{L} = \langle \mathcal{T}, \preceq \rangle$. The associated value $c \in \mathcal{T}$ indicates to which *certainty* a predicate is $true$.

Syntax. From a syntax point of view, as the description logic component, we consider $\mathcal{L}$-$\mathcal{ALC}$. We then extend the DL component with rules, where the management of uncertainty is based on so-called annotation terms. Annotation terms and the form of the rules are like those in the Generalized Annotated Logic Programming framework of Kifer and Subrahmanian [33], except that now DL atoms and roles may appear in a rule body.

Let us define an *annotation function* of arity n to be a total, *monotone* and computable function [4] $f\colon (\mathcal{T})^n \to \mathcal{T}$. Assume a new alphabet of *annotation variables*, which will denote a value in $\mathcal{T}$ and can only appear in so-called *annotation terms*. An *annotation term*, λ, is defined inductively: (i) as a constant $c \in \mathcal{T}$, or as an annotation variable (ν), or (ii) is of the form $f(\lambda_1, \ldots, \lambda_n)$, where f is an n-ary annotation function and all λ_i are annotation terms. Annotation terms are supposed to denote elements of $\mathcal{T}$. Now, let A be an atom and λ an annotation term. A $a-$*atom*, denoted $\bar{A}$, is of the form $A\colon \lambda$. The intended meaning is that "the certainty of A is greater or equal to λ". A $a-$*logic program* (or, simply $a-$program), $\bar{\mathcal{P}}$, is a finite set of $a-$*rules* of the form

$$H\colon \lambda \leftarrow B_1\colon \lambda_1, \ldots, B_n\colon \lambda_n \ ,$$

where $H\colon \lambda$ and all $B_1\colon \lambda_1$ are $a-$atoms. Finally, a *Annotated Description Logic Program* (ADLP), denoted $\bar{\mathcal{DP}}$, is a pair $\langle K, \bar{\mathcal{P}} \rangle$, where K is a $\mathcal{L}$-$\mathcal{ALC}$ knowldege base and $\bar{\mathcal{P}}$ is a $a-$program.

Semantics. In order to avoid straightforward repetition, if not stated otherwise, definitions related to $a-$logic programs, parallels those for classical logic programs. Furthermore, in grounding a $a-$atom $A\colon \lambda$, we assume that the annotation term λ is grounded as well, *i.e.* annotation variables and constants are replaced with values in $\mathcal{T}$ and annotation terms of the form $f(\lambda_1, \ldots, \lambda_n)$ are replaced with the result of the computation of $f(\lambda_1, \ldots, \lambda_n)$. A is grounded with the (individual) constants appearing in $\bar{\mathcal{DP}}$. Note that a grounded $a-$program $\bar{\mathcal{P}}$ may contain an infinite number of rules due to the grounding of annotation terms. For a grounded $a-$program $\bar{\mathcal{P}}$, the Herbrand base $\mathcal{B}_{\bar{\mathcal{P}}}$ is the set of ground non-DL atoms A that can be constructed using the predicate symbols in $\bar{\mathcal{P}}$ (annotations terms are not considered in the grounding of A). With $\texttt{ground}(\bar{\mathcal{P}})$ we denote the set of all ground rules of $\bar{\mathcal{P}}$.

An $a-$*interpretation* I of a grounded $a-$program $\bar{\mathcal{P}}$ is a function $I\colon \mathcal{B}_{\bar{\mathcal{P}}} \to \mathcal{T}$. We say I is a *model* of a ground $a-$atom $A\colon \lambda$ w.r.t. K, denoted $I \models_K A\colon \lambda$, iff

- $I(A) \succeq \lambda$, if $A \in \mathcal{B}_{\mathcal{P}}$ (A is a non-DL atom);
- $K \models_{\mathcal{L}} \langle A, \lambda \rangle$ if A is a DL atom.

The notion of model is then straightforwardly extended to $a-$rules, $a-$programs $\bar{\mathcal{P}}$ and ADLPs $\bar{\mathcal{DP}}$. The notion of entailment, $\bar{\mathcal{DP}} \models A\colon \lambda$, is as usual.

We say that $I \preceq I'$ iff for all atoms $A \in \mathcal{B}_{\bar{\mathcal{P}}}$, $I(A) \preceq I'(A)$. Like for the crisp case (see Proposition 1) we have:

[4] The result of f is computable in a finite amount of time.

Proposition 4 *Let $\bar{\mathcal{D}P} = \langle K, \bar{\mathcal{P}} \rangle$ be a ADLP. If $\bar{\mathcal{D}P}$ is satisfiable, then there exists a unique model $M_{\bar{\mathcal{D}P}}$ of $\bar{\mathcal{D}P}$ such that $M_{\bar{\mathcal{D}P}} \preceq I$ for all models I of $\bar{\mathcal{D}P}$. Furthermore, for any ground $a-$atom $\bar{A}$, $\bar{\mathcal{D}P} \models \bar{A}$ iff $M_{\bar{\mathcal{D}P}} \models_K \bar{A}$.*

Like in [33], the minimal model coincides with the least fixed-point of the following monotone operator. Let $\bar{\mathcal{D}P} = \langle K, \bar{\mathcal{P}} \rangle$ be a ADLP. Define the operator $T_{\bar{\mathcal{D}P}}$ on interpretations as follows: for every interpretation I, for all $A \in \mathcal{B}_{\bar{\mathcal{P}}}$ let

$$T_{\bar{\mathcal{D}P}}(I)(A) = \bigoplus \{ \lambda \mid A{:}\,\lambda \leftarrow B_1{:}\,\lambda_1, \ldots, B_n{:}\,\lambda_n \in \bar{\mathcal{P}}, I \models_K B_i{:}\,\lambda_i, \text{ for all } i \} \ .$$

Note however, that $T_{\bar{\mathcal{D}P}}$ is not continuous in general. In fact the grounded $a-$program (over $\mathcal{L}_{[0,1]}$), containing all ground instances of the rules

$$\begin{array}{rcl} A{:}\,0. & & \\ A{:}\,\dfrac{\nu+1}{2} & \leftarrow & A{:}\,\nu \\ B{:}\,1 & \leftarrow & A{:}\,1 \end{array}$$

has unique minimal model $I(A) = I(B) = 1$, which is obtained after $\omega + 1$ $T_{\bar{\mathcal{D}P}}$ iterations starting from $I_\perp$ (the interpretation assigning 0 to all atoms), where ω is the first limit ordinal. However, under the assumption that the certainty lattice $\mathcal{L} = \langle \mathcal{T}, \preceq \rangle$ is finite, the minimal model can be computed after at most $\mathcal{O}(|\bar{\mathcal{D}P}||\mathcal{T}|)$ steps (any rule can be applied at most $\mathcal{O}(|\mathcal{T}|)$ times). In particular, in the example above, in case we consider two digit precision of $[0, 1]$ numbers, the minimal model is computed after at most 300 (precisly, 10) applications of the $T_{\bar{\mathcal{D}P}}$ iterations starting from $I_\perp$.

Like for the crisp case, from an implementation point of view, given $\mathcal{D}P = \langle K, \mathcal{P} \rangle$, we can compute the set of entailment assertions $F = \{\langle A, \lambda \rangle{:}\, K \models_{\mathcal{L}} \langle A, \lambda \rangle,$ where A is a DL atom$\}$ first and then add them as facts to $\bar{\mathcal{P}}$. Then, we can rely on theorem proving for $\bar{\mathcal{P}}$ either using a usual bottoum-up computation or a top-down computation like [9, 64].

Example 2 *Consider an insurance company, which has information about its customers used to determine the risk coefficient of each customer. The company has: (i) data grouped into a set of facts* $\{$`Experience(john)`$:0.7.$, `Risk(john)`$:0.5.$, `Sport_car(john)`$:0.8.\}$*; and (ii) a set of rules (for ease we omit the terminological axioms about the involved concepts).*

$$\begin{array}{lcl} \texttt{Good\_driver(x)}{:}\,\min(\nu, \nu'/2) & \leftarrow & \texttt{Experience(x)}{:}\,\nu, \texttt{Risk(x)}{:}\,\nu' \\ \texttt{Risk(x)}{:}\,\nu \cdot 0.8 & \leftarrow & \texttt{Young(x)}{:}\,\nu \\ \texttt{Risk(x)}{:}\,\nu \cdot 0.8 & \leftarrow & \texttt{Sport\_car(x)}{:}\,\nu \\ \texttt{Risk(x)}{:}\,\min(\nu, \nu'/2) & \leftarrow & \texttt{Experience(x)}{:}\,\nu, \texttt{Good\_driver(x)}{:}\,\nu' \end{array}$$

It turns out that

$$\begin{array}{l} \texttt{Risk(john)}{:}\,0.64 \\ \texttt{Sport\_car(john)}{:}\,0.8 \\ \texttt{Young(john)}{:}\,0 \\ \texttt{Good\_driver(john)}{:}\,0.32 \\ \texttt{Experience(john)}{:}\,0.7 \end{array}$$

are entailed from the above facts and the rules, which establish that `john`*'s degree of* `Risk` *is greater or equal than* 0.64. □

5 Related work

While the combination of DLs with LPs is not new, to best of our knowledge, the generality of DLPs and integration of the management of uncertainty as well, has not been investigated yet.

Concerning ADLPs, the DL component mainly relies on the work of Straccia [62]. Concerning the rule component, as anticipated, the Generalized Annotated Logic Programming framework of Kifer and Subrahmanian [33] inspires it.

We recall that concerning the combination of DLs with LPs, they do not consider function symbols and, the works can roughly be divided into (*i*) hybrid approaches, which use description logics to specify structural constraints in the bodies of logic programs rules; and (*ii*) approaches that reduce description logic inference to logic programming. The basic idea behind (*i*) is to combine the semantic and computational strengths of the two systems, while the main rationale of (*ii*) is to use powerful logic programming technology for inference in description logics.

In the former case fall approaches like [12, 27, 37]. In particular, [12] combines plain Datalog (no disjunction and negation) with the description logic $\mathcal{ALC}$, where the integration lies in using concepts from the terminological component as constraints in rule bodies. It also presents a technique for answering conjunctive queries (existentially quantified conjunctions of atoms) with such constraints, where SLD resolution is integrated with an inference method for $\mathcal{ALC}$. More related to our approach is [37], which combines Horn rules with the description logic $\mathcal{ALCNR}$, where concepts and roles may appear in the body of Horn rules. Like in [12], [37] devices an SLD resolution integrated with an inference method for $\mathcal{ALCNR}$. Finally, [27] is similar to [37], except that the atoms of a rule refer to OWL classes and properties rather than to $\mathcal{ALCNR}$.

In the latter case (reducing description logic inference to function-free logic programming) fall approaches like [1, 21, 23, 44, 29, 45, 46, 65, 68]. We remark that (*i*) [68] reduces knowledge bases in the DL $\mathcal{ALCN}$ ($\mathcal{ALC}$ with number restrictions) to open logic programs; (*i*) [1, 65] reduce reasoning in the DL $\mathcal{ALCQI}$ ($\mathcal{ALC}$ with qualified number restrictions and inverse roles) to query answering from answer sets of normal logic programs; (*iii*) [21] shows how a subset of the DL $\mathcal{SHOIQ}$ can be reduced to a subset of Horn programs (positive normal programs); (*iv*) [23] reduces the DL $\mathcal{SHIF}$ with transitive role closure to disjunctive logic programs (in [61] we use a similar reduction); and [44, 29, 45, 46] reduces reasoning in the DL $\mathcal{SHIQ}^-$ to reasoning in disjunctive Datalog programs by translating DL expressions into FOL clauses, saturates the clauses by resolution and translates the saturated clauses to disjunctive Datalog programs.

Finally, not in the above classification fall approaches like [14, 15, 70].

Roughly, [14, 15] combines the DL $\mathcal{SHOIN}(D)$, which is $\mathcal{SHOIN}$ with concrete domains (*e.g.* integers, reals), with normal logic programs. The main characteristics of [14] lies in the use of a weaker semantics to be used to get decidable reasoning. Indeed, concept and role predicates appearing in rules can be managed as system calls to a DL reasoner. Our approach is inspired on this work, but considered uncertainty components. Finally, [70] is based on *F-logic* [32].

6 Conclusion

We have presented a DL framework for the management of uncertain information. Our main feature is that a sentence is not just true or false like in classical DLs, but *certain* to some degree, where the certainty value is taken from a certainty lattice. Syntax, semantics and a sound and complete tableaux algorithm for reasoning in it has been presented. We also extended it with a rule component based on annotated logic programs, forming the class of annotated description logic programs. We also discussed how to reason in it.

7 Acknowledgments

I'm in debt with the anonymous reviewers who provided useful comments to correct and improve an early version of this paper.

References

[1] Guray Alsaç and Chitta Baral. Reasoning in description logics using declarative logic programming. Technical report, Department of Computer Science and Engineering, Arizona State University, USA, 1997.

[2] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[3] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[4] Nuel D. Belnap. A useful four-valued logic. In Gunnar Epstein and J. Michael Dunn, editors, *Modern uses of multiple-valued logic*, pages 5–37. Reidel, Dordrecht, NL, 1977.

[5] Harold Boley, Said Tabet, and Gerd Wagner. Design rationale of RuleML: A markup language for semantic web rules. In *Proceedings of the 1st Semantic Web Working Symposium (SWWS-01)*, pages 105–113, Stanford, 2001.

[6] P. Bonatti and A. Tettamanzi. Some complexity results on fuzzy description logics. In A. Petrosino V. Di Gesù, F. Masulli, editor, *WILF 2003 International Workshop on Fuzzy Logic and Applications*, LNCS 2955, Berlin, 2004. Springer Verlag.

[7] G. Bordogna, P Carrara, and G. Pasi. Query term weights as constraints in fuzzy information retrieval. *Information Processing and Management*, 27(1):15–26, 1991.

[8] Carlos Viegas Damásio, J. Medina, and M. Ojeda Aciego. Sorted multi-adjoint logic programs: Termination results and applications. In *Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA-04)*, number 3229 in Lecture Notes in Computer Science, pages 252–265. Springer Verlag, 2004.

[9] Carlos Viegas Damásio, J. Medina, and M. Ojeda Aciego. A tabulation proof procedure for residuated logic programming. In *Proceedings of the 6th European Conference on Artificial Intelligence (ECAI-04)*, 2004.

[10] S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, S. Rajagopalan, A. Tomkins, J.A. Tomlin, and J.Y. Zien. SemTag: and Seeker: Bootstrapping the semantic web via automated semantic annotation. In *The Twelfth International World Wide Web Conference (WWW-03)*, pages –, Budapest, Hungary, 2003.

[11] F. M. Donini, M. Lenzerini, D. Nardi, W. Nutt, and A. Schaerf. An epistemic operator for description logics. *Artificial Intelligence*, 100(1-2):225–274, 1998.

[12] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. AL-log: Integrating datalog and description logics. *Journal of Intelligent Information Systems*, 10(3):227–252, 1998.

[13] Didier Dubois, Jérome Lang, and Henri Prade. Towards possibilistic logic programming. In *Proc. of the 8th Int. Conf. on Logic Programming (ICLP-91)*, pages 581–595. The MIT Press, 1991.

[14] Thomas Eiter, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Combining answer set programming with description logics for the semantic web. In *Proceedings of the 9th International Conference on Principles of Knowledge Representation and Reasoning (KR-04)*. AAAI Press, 2004.

[15] Thomas Eiter, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Well-founded semantics for description logic programs in the semantic web. In *Proceedings RuleML 2004 Workshop, International Semantic Web Conference*, number 3323 in Lecture Notes in Computer Science, pages 81–97. Springer Verlag, 2004.

[16] Andrei Lopatenko Enrico Franconi, Gabriel Kuper and Luciano Serafini. A robust logical and computational characterisation of peer-to-peer database systems. In *roceedings of the VLDB International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P-03)*, 2004.

[17] Pan *et al.* Specification of coordination of rule and ontology languages. Technical report, Knowledgeweb Network of Excellence, EU-IST-2004-507482, 2004. Deliverable D2.5.1.

[18] Enrico Franconi and Sergio Tessaris. Rules and queries with ontologies: a unified logical framework. In *Workshop on Principles and Practice of Semantic Web Reasoning (PPSWR-04)*, 2004.

[19] Th. Gevers and A.W.M. Smeulders. Content-based image retrieval: An overview. In G. Medioni and S. B. Kang, editors, *Emerging Topics in Computer Vision*, page To appear. Prentice Hall, 2004.

[20] T. Rosalba Giugno and Thomas Lukasiewicz. P-SHOQ(D): A probabilistic extension of SHOQ(D) for probabilistic ontologies in the semantic web. In *Proceedings of the 8th European Conference on Logics in Artificial Intelligence (JELIA'02)*, number 2424 in Lecture Notes in Artificial Intelligence, pages –. Springer-Verlag, 2002.

[21] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: combining logic programs with description logic. In *Proceedings of the twelfth international conference on World Wide Web*, pages 48–57. ACM Press, 2003.

[22] Jochen Heinsohn. Probabilistic description logics. In R. Lopez de Mantara and D. Pool, editors, *Proceedings of the 10th Conference on Uncertainty in Artificila Intelligence*, pages 311–318, 1994.

[23] Stijn Heymans and Dirk Vermeir. Integrating description logics and answer set programming. In *Principles and Practice of Semantic Web Reasoning (PPSWR-03)*, number 2901 in Lecture Notes in Computer Science, pages 146–159, Mumbai, India, 2003. Springer Verlag.

[24] Steffen Hölldobler, Hans-Peter Störr, and Tran Dinh Khang. The subsumption problem of the fuzzy description logic ALC_{FH}. In *Proceedings of the 10th International Conference on Information Processing and Managment of Uncertainty in Knowledge-Based Systems, (IPMU-04)*, 2004.

[25] Bernhard Hollunder. An alternative proof method for possibilistic logic and its application to terminological logics. In *10th Annual Conference on Uncertainty in Artificial Intelligence*, pages 327–335, Seattle, Washington, 1994. Morgan Kaufmann.

[26] Ian Horrocks and Peter F. Patel-Schneider. Three theses of representation in the semantic web. In *Proceedings of the 12th International Conference on World Wide Web*, pages 39–47. ACM Press, 2003.

[27] Ian Horrocks and Peter F. Patel-Schneider. A proposal for an OWL rules language. In *Proc. of the Thirteenth International World Wide Web Conference (WWW-04)*. ACM, 2004.

[28] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.

[29] U. Hustadt, B. Motik, and U. Sattler. Reasoning in description logics with a concrete domain in the framework of resolution. In *Proceedings of the 6th European Conference on Artificial Intelligence (ECAI-04)*, 2004.

[30] Manfred Jäger. Probabilistic reasoning in terminological logics. In *Proceedings of KR-94, 5-th International Conference on Principles of Knowledge Representation and Reasoning*, pages 305–316, Bonn, FRG, 1994.

[31] E.E. Kerre, R.B. Zenner, and R.M. De Caluwe. The use of fuzzy set theory in information retrieval and databases: a survey. *Journal of the American Society for Information Science*, 37(5):341–345, 1986.

[32] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of Object-Oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, 1995.

[33] Michael Kifer and V.S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming*, 12:335–367, 1992.

[34] Daphne Koller, Alon Levy, and Avi Pfeffer. P-CLASSIC: A tractable probabilistic description logic. In *Proc. of the 14th Nat. Conf. on Artificial Intelligence (AAAI-97)*, pages 390–397, 1997.

[35] Donald H. Kraft and Duncan Buel. Fuzzy sets and generalised boolean retrieval systems. *Int. J. Man-Machine Studies*, 19:45–56, 1983.

[36] Laks V.S. Lakshmanan and Nematollaah Shiri. A parametric approach to deductive databases with uncertainty. *IEEE Transactions on Knowledge and Data Engineering*, 13(4):554–570, 2001.

[37] Alon Y. Levy and Marie-Christine Rousset. Combining horn rules and description logics in CARIN. *Artificial Intelligence*, 104:165–209, 1998.

[38] John W. Lloyd. *Foundations of Logic Programming*. Springer, Heidelberg, RG, 1987.

[39] Yann Loyer and Umberto Straccia. The approximate well-founded semantics for logic programs with uncertainty. In *28th International Symposium on Mathematical Foundations of Computer Science (MFCS-2003)*, number 2747 in Lecture Notes in Computer Science, pages 541–550, Bratislava, Slovak Republic, 2003. Springer-Verlag.

[40] Thomas Lukasiewicz. Probabilistic logic programming. In *Proc. of the 13th European Conf. on Artificial Intelligence (ECAI-98)*, pages 388–392, Brighton (England), August 1998.

[41] C. Lutz, F. Wolter, and M. Zakharyaschev. A tableau algorithm for reasoning about concepts and similarity. In *Proceedings of the Twelfth International Conference on Automated Reasoning with Analytic Tableaux and Related Methods TABLEAUX 2003*, number 2796 in LNAI, Rome,Italy, 2003. Springer.

[42] Cristinel Mateis. Quantitative disjunctive logic programming: Semantics and computation. *AI Communications*, 13:225–248, 2000.

[43] Carlo Meghini, Fabrizio Sebastiani, and Umberto Straccia. A model of multimedia information retrieval. *Journal of the ACM*, 48(5):909–970, 2001.

[44] B. Motik, U. Sattler, and U. Hustadt. Reducing $\mathcal{SHIQ}^-$ description logic to disjunctive datalog programs. In *Proceedings of the 9th International Conference on Principles of Knowledge Representation and Reasoning (KR-04)*. AAAI Press, 2004.

[45] B. Motik, R. Volz, and A. Maedche. Optimizing query answering in description logics using disjunctive deductive databases. In *10th International Workshop on Knowledge Representation meets Databases (KRDB-03)*, pages 39–50, 2003.

[46] Boris Motik, Ulrike Sattler, and Rudi Studer. Query answering for OWL-DL with rules. In *Proceedings of the International Semantic Web Conference*, number 3298 in Lecture Notes in Computer Science, pages 549–563. Springer Verlag, 2004.

[47] Bernhard Nebel. *Reasoning and revision in hybrid representation systems*. Springer, Heidelberg, FRG, 1990.

[48] C.V. Negoita and P. Flondor. On fuzziness in information retrieval. *Int. J. Man-Machine Studies*, 8:711–716, 1976.

[49] Raymond Ng and V.S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1993.

[50] Peter F. Patel-Schneider. A four-valued semantics for terminological logics. *Artificial Intelligence*, 38:319–351, 1989.

[51] Dan Roth. On the hardness of approximate reasoning. *Artificial Intelligence Journal*, 82:273–302, 1996.

[52] D. Sánchez and G.B. Tettamanzi. Generalizing quantification in fuzzy description logics. In *Proceedings 8th Fuzzy Days in Dortmund*, 2004.

[53] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48:1–26, 1991.

[54] Fabrizio Sebastiani. A probabilistic terminological logic for modelling information retrieval. In *Proceedings of SIGIR-94, 17th ACM International Conference on Research and Development in Information Retrieval*, pages 122–130, Dublin, IRL, 1994. Published by Springer Verlag, Heidelberg, FRG.

[55] Umberto Straccia. A sequent calculus for reasoning in four-valued description logics. In *Proc. of the Int. Conf. on Analytic Tableaux and Related Methods (TABLEAUX-97)*, number 1227 in Lecture Notes in Artificial Intelligence, pages 343–357, Pont-à-Mousson, France, 1997.

[56] Umberto Straccia. A fuzzy description logic. In *Proc. of the 15th Nat. Conf. on Artificial Intelligence (AAAI-98)*, pages 594–599, Madison, USA, 1998.

[57] Umberto Straccia. A framework for the retrieval of multimedia objects based on four-valued fuzzy description logics. In F. Crestani and Gabriella Pasi, editors, *Soft Computing in Information Retrieval: Techniques and Applications*, pages 332–357. Physica Verlag (Springer Verlag), Heidelberg, Germany, 2000.

[58] Umberto Straccia. Reasoning within fuzzy description logics. *Journal of Artificial Intelligence Research*, 14:137–166, 2001.

[59] Umberto Straccia. Towards a fuzzy description logic for the semantic web. Technical Report 2004-TR-64, Istituto di Scienza e Tecnologie dell'Informazione, Consiglio Nazionale delle Ricerche, Pisa, Italy, 2004.

[60] Umberto Straccia. Transforming fuzzy description logics into classical description logics. In *Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA-04)*, number 3229 in Lecture Notes in Computer Science, pages 385–399, Lisbon, Portugal, 2004. Springer Verlag.

[61] Umberto Straccia. Uncertainty in description logic programs. Technical Report ISTI-2004-TR-01, Istituto di Scienza e Tecnologie dell'Informazione, Consiglio Nazionale delle Ricerche, Pisa, Italy, 2004.

[62] Umberto Straccia. Uncertainty in description logics: a lattice-based approach. In *Proceedings of the 10th International Conference on Information Processing and Managment of Uncertainty in Knowledge-Based Systems, (IPMU-04)*, pages 251–258, 2004.

[63] Umberto Straccia. Fuzzy description logics with concrete domains. Technical Report 2005-TR-03, Istituto di Scienza e Tecnologie dell'Informazione, Consiglio Nazionale delle Ricerche, Pisa, Italy, 2005.

[64] Umberto Straccia. Uncertainty management in logic programming: Simple and effective top-down query answering. In *9th International Conference on*

Knowledge-Based & Intelligent Information & Engineering Systems (KES-05), Lecture Notes in Computer Science, pages –, Melbourne, Australia, 2005. Springer Verlag.

[65] Terrance Swift. Deduction in ontologies via ASP. In *Proceedings of the 7th International Conference in Logic Programming and Nonmonotonic Reasoning (LPNMR-04)*, number 2923 in Lecture Notes in Artificial Intelligence, pages 275–288, Fort Lauderdale, FL, USA, 2004. Springer Verlag.

[66] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, (5):285–309, 1955.

[67] C. Tresp and R. Molitor. A description logic for vague knowledge. In *Proc. of the 13th European Conf. on Artificial Intelligence (ECAI-98)*, Brighton (England), August 1998.

[68] K. van Belleghem, M. Denecker, and D. de Schreye. A strong correspondence between description logics and open logic programs. In *Proc. of the 13th Int. Conf. on Logic Programming (ICLP-97)*, pages 346–360. The MIT Press, 1997.

[69] Peter Vojtáš. Fuzzy logic programming. *Fuzzy Sets and Systems*, 124:361–370, 2004.

[70] Guizhen Yang, Michael Kifer, and Chang Zhao. Flora-2: A rule-based knowledge representation and inference infrastructure for the semantic web. In *Second International Conference on Ontologies, Databases and Applications of Semantics (ODBASE-03)*, number 2888 in Lecture Notes in Computer Science, pages 671–688, Catania, Sicily, Italy, 2003. Springer Verlag.

[71] John Yen. Generalizing term subsumption languages to fuzzy logic. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 472–477, Sydney, Australia, 1991.