Uncertainty and Description Logic Programs: A Proposal for Expressing Rules and Uncertainty on Top of Ontologies

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

Abstract

Rule-based and object-oriented techniques are rapidly making their way into the infrastructure for representing and reasoning about the Semantic Web and combining these two paradigms emerges as an important objective. We present a new family of representation languages, which extents existing language families for the Semantic Web: namely Description Logic Programs (DLPs) and DLPs with uncertainty ($\mu$DLPs). The former combine the expressive power of description logics (which capture the meaning of the most popular features of structured representation of knowledge) and disjunctive logic programs (powerful rule-based representation languages). The latter are DLPs in which the management of uncertainty is considered as well. We show that $\mu$DLPs may be applied in the context of distributed information search in the Semantic Web, where the representation of the inherent uncertainty of the relationships among resource ontologies, to which an automated agent has access to, is required.

Category: F.4.1: Mathematical Logic and Formal Languages: Mathematical Logic: [Logic and constraint programming]

Category: I.2.3: Artificial Intelligence: Deduction and Theorem Proving: [Logic programming]

Terms: Theory

Keywords: Description Logics, Logic programs, uncertainty, Semantic Web

1 Introduction

The Semantic Web\(^1\) \cite{7} is widely regarded as the next step in the evolution of the World Wide Web. It aims at enhancing content on the World Wide Web with machine-readable meta-data, that should support agents (machines or human users) to richer discovery, data integration, navigation, and automation of tasks. Ontologies \cite{34} play

\(^1\)www.semanticweb.org
a key role in the Semantic Web and major effort has been put by the Semantic Web community into this issue. Ontologies provide a source of shared and precisely defined terms that can be used in such meta-data. Informally, an ontology consists typically of a hierarchical description of important concepts in a particular domain, along with the description of the properties (of the instances of) each concept. That is, an ontology defines a representation of a shared conceptualization of a particular world (a conceptual schema). The Semantic Web is envisioned to contain several languages in which languages of increasing power are layered one on top of the other [42, 44]. The OWL Web Ontology Language [44] is currently the highest layer of sufficient maturity. OWL has three increasingly expressive sub-languages, namely OWL Lite, OWL DL and OWL Full, where OWL DL corresponds basically to DAML+OIL [41]. OWL Lite and OWL DL are essentially very expressive Description Logics (DLs) [4] with an RDF syntax (see, e.g. [44, 39, 41, 42, 78]). DLs provide a simple well-established Tarski-style declarative semantics to capture the meaning of the most popular features of structured representation of knowledge.

Although OWL adds considerable expressive power to the Semantic Web it does have expressive limitations, particularly with respect to what can be said about properties. An intuitive way to overcome to these limitations is to extend current ontology languages with rules, i.e. to allow for building rules that use terms specified in ontologies. A first effort in this direction is RuleML [9, 8], fostering an XML-based markup language for rules and rule-based systems, while the OWL Rules Language [43] is a first proposal for extending OWL by Horn clause rules.

As first contribution of this paper, we propose, towards the integration of rules and ontologies in the Semantic Web, a framework for the combination of Disjunctive Logic Programs (LPs) under the answer set semantics [30, 56] with description logics. In particular, in this context the main contributions can be summarized as follows: we introduce the notion of Description Logic Programs (DLPs), which consists of a knowledge base (ontology) in description logic and a finite set of rules of disjunctive logic programs with explicit and negation as failure. Such rules are similar to the usual rules in logic programming, but may have

1. disjunctions in the head of a rule;
2. negation as failure in the head and the body of a rule;
3. explicit negation in the head and the body of a rule;
4. terms defined in the ontology in the head and the body of a rule and, thus, rules may contribute to the “intentional” definition of terms belonging to the ontology;
5. the body or the head of a rule may be empty and, thus, allows us to express constraints as well as facts.

The combination of DLs with LPs is not new (see, e.g. [20, 25, 33, 36, 43, 55]), however these differ significantly from our work, as we discuss in more detail on the related work later on. Rather than to propose another particular alternative language, our principal goal is to propose a framework for a principled integration of both representation paradigms based on description logic and logic programming, to which alternative formalizations may be compared in terms of computational complexity and/or expressive power.

---

2See the DL community home page http://dl.kr.org/.
As the Semantic Web grows in importance, resources will probably start to export their data and/or service descriptions according to some chosen ontology. Resource discovery, searching and resource integration (see, e.g. [3, 102, 54]), thus, become major challenges to information access for which accurate automated tools are desired.

As building an ontology is a time consuming and expensive process, the added value of both ontology-based document and/or service annotation and access to (integration of/search in) ontology-based resources should clearly compensate the enormous labour to construct it. Clearly, while the construction of an ontology and the description of Web services may accepted to be manual, the semantic-annotation of Web documents and the integration/access of heterogeneous resources should be automatic in the long run and the large scale. This is, without doubt a challenging and long term issue.

Towards this end, we need a suitable representation language and likely, it should be able to deal with the management of the inherent uncertainty in the above tasks, as the following case exemplifies.

Let us assume that an agent $A$ has to satisfy an information need $Q_A$ expressed in a query language $L_A$, whose basic terms belong to an ontology $O_A$, defined using the ontology language $O_A$. Assume that there are a large amount of Semantic Web resources $\mathcal{S} = \{S_1, \ldots, S_n\}$ accessible to $A$, where each resource $S_i$ provides access to its objects by having its own ontology $O_i$, ontology language $O_i$ and query language $L_i$. Then the roughly an agent has to perform the following steps:

1. the agent has to select a subset of relevant (or, most promising) resources he is aware of, $\mathcal{S}' \subseteq \mathcal{S}$, as it is not reasonable to assume to access to and query all sources (resource selection);
2. for every selected resource $S_i \in \mathcal{S}'$ the agent has to reformulate its information need $Q_A$ into the query language $L_i$ provided by the resource (schema mapping);
3. the results from the selected sources have to be merged together (data fusion).

That is, an agent must know where to search, how to query different resources, and how to combine information from diverse resources. The above problems have already been addressed recently, envisaging fully automatic processes, in the context of Distributed Information Retrieval and in the database area. Investigations addressed the problem both globally [3, 6, 71, 96, 101], as well as locally in terms of its sub-tasks - resource selection [11, 28, 62, 74]; schema mapping [69, 70, 72]; data-fusion [12, 22, 75] and all proposed solutions require the management of the inherent uncertainty in the above tasks.

Our second contribution of this paper is the proposal of a framework, in which we extend DLPs towards the management of uncertainty to support reasoning agents in the Semantic Web. In particular, we propose to augment DLPs with annotation terms indicating to which degree an atom is certain, by relying on the work pioneered by [50]. As for DLPs, we define abstract syntax, semantics and discuss computational issues. The extension of DLPs to the management of uncertainty, to the best of our knowledge, has not been investigated yet.

We proceed as follows. We first briefly introduce the main notions related to description logics and disjunctive logic programs, and then show how both can be integrated, defining Description Logic Programs (DLPs). We then finally extend DLPs with the management of uncertainty, present related work and conclude.
2 Preliminaries

The OWL ontology languages OWL Lite and OWL DL are very close to expressive DLs, i.e. $\text{SHIF}(D)$ and $\text{SHOIN}(D)$, respectively [42]. For the sake of ease our presentation, we are not going to consider OWL DL as a whole, but restrict ourself to a significant subset of it. The specific DL we consider is $\text{ALC}$ [79], a significant representative of DLs. $\text{ALC}$ is sufficiently expressive to illustrate the main concepts introduced in this paper. We then recall disjunctive logic programs under the answer set semantics.

2.1 Description logics

2.1.1 Syntax.

We first describe the syntax of $\text{ALC}$. Consider two alphabets of symbols, for concepts names (denoted $A$) and for roles names (denoted $R$). A concept (denoted $C$ or $D$) of the language $\text{ALC}$ is built inductively from concept names $A$ and role names $R$ according to the following syntax rule:

- $C, D \rightarrow \top$ (top concept)
- $C, D \rightarrow \bot$ (bottom concept)
- $C, D \rightarrow A$ (concept name)
- $C, D \rightarrow C \sqcap D$ (concept conjunction)
- $C, D \rightarrow C \sqcup D$ (concept disjunction)
- $C, D \rightarrow \neg C$ (concept negation)
- $C, D \rightarrow \forall R.C$ (universal quantification)
- $C, D \rightarrow \exists R.C$ (existential quantification).

A terminology, $T$, is a finite set of concept inclusions or role inclusions, called terminological axioms, $\tau$, where given two concepts $C$ and $D$, and two role names $R$ and $R'$, a terminological axiom is an expression of the form $C \sqsubseteq D$ (D subsumes C) or of the form $R \sqsubseteq R'$ (R' subsumes R). We write $C = D$ (resp. $R = R'$) is place of the pair of axioms $C \sqsubseteq D$ and $D \sqsubseteq C$ (resp. $R \sqsubseteq R'$ and $R' \sqsubseteq R$).

2.1.2 Semantics.

An interpretation $I$ is a pair $I = (\Delta I, \cdot I)$ consisting of a non empty set $\Delta I$ (called the domain) and of an interpretation function $\cdot I$ mapping concepts names into subsets of $\Delta I$ and roles names into subsets of $\Delta I \times \Delta I$. The interpretation of complex concepts is defined inductively as usual:

- $\top I = \Delta I$
- $\bot I = \emptyset$
- $(C \sqcap D) I = C I \cap D I$
- $(C \sqcup D) I = C I \cup D I$
- $(-C) I = \Delta I \setminus C I$
- $(\forall R.C) I = \{d \in \Delta I \mid \forall d', (d, d') \notin R I \text{ or } d' \in C I\}$
- $(\exists R.C) I = \{d \in \Delta I \mid \exists d', (d, d') \in R I \text{ and } d' \in C I\}$.

Metavariables may have a subscript or a superscript.
### Table 1: Some OWL DL constructs and their DL counterpart

<table>
<thead>
<tr>
<th>OWL DL Abstract Syntax</th>
<th>DL Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Descriptions(C)</td>
<td>A</td>
</tr>
<tr>
<td>A (URI reference)</td>
<td>A</td>
</tr>
<tr>
<td>owl:Thing</td>
<td>⊤</td>
</tr>
<tr>
<td>owl:Nothing</td>
<td>⊥</td>
</tr>
<tr>
<td>intersectionOf(C₁ ... Cₙ)</td>
<td>C₁ ∩ C₂ ∩ ... ∩ Cₙ</td>
</tr>
<tr>
<td>unionOf(C₁ ... Cₙ)</td>
<td>C₁ ∪ C₂ ∪ ... ∪ Cₙ</td>
</tr>
<tr>
<td>complementOf(C)</td>
<td>¬C</td>
</tr>
<tr>
<td>restriction(R someValuesFrom(C))</td>
<td>∃R.C</td>
</tr>
<tr>
<td>restriction(R allValuesFrom(C))</td>
<td>∀R.C</td>
</tr>
<tr>
<td>Object properties (R)</td>
<td>R</td>
</tr>
<tr>
<td>R (URI reference)</td>
<td>R</td>
</tr>
<tr>
<td>Axioms</td>
<td></td>
</tr>
<tr>
<td>Class(A partial C₁ ... Cₙ)</td>
<td>A ⊑ C₁ ∩ C₂ ∩ ... ∩ Cₙ</td>
</tr>
<tr>
<td>Class(A complete C₁ ... Cₙ)</td>
<td>A = C₁ ∩ C₂ ∩ ... ∩ Cₙ</td>
</tr>
<tr>
<td>SubClassOf(C₁ C₂)</td>
<td>C₁ ⊆ C₂</td>
</tr>
<tr>
<td>EquivalentClasses(C₁ ... Cₙ)</td>
<td>C₁ = C₂ = ... = Cₙ</td>
</tr>
<tr>
<td>DisjointClasses(C₁ ... Cₙ)</td>
<td>Cᵢ ∩ Cⱼ = ⊥, i ≠ j</td>
</tr>
<tr>
<td>ObjectProperty(R super(R₁) ... super(Rₙ))</td>
<td>R ⊑ Rᵢ</td>
</tr>
<tr>
<td>domain(C₁) ... domain(Cₘ)</td>
<td>∃R. ⊑ Cᵢ</td>
</tr>
<tr>
<td>range(C₁) ... range(Cₙ)</td>
<td>⊤ ⊑ ∀R.Cᵢ</td>
</tr>
<tr>
<td>SubPropertyOf(R₁ R₂)</td>
<td>R₁ ⊑ R₂</td>
</tr>
<tr>
<td>EquivalentProperties(R₁ R₂ ... Rₙ)</td>
<td>R₁ = R₂ = ... = Rₙ</td>
</tr>
</tbody>
</table>

An interpretation \( I \) satisfies a terminological axiom \( C \sqsubseteq D \) (resp. \( R \sqsubseteq R' \)) iff \( C^I \subseteq D^I \) (resp. \( R^I \subseteq R'^I \)), while \( I \) satisfies (is a model of) a terminology \( T \) iff \( I \) satisfies each element in \( T \).

The syntax we have presented is in typical DL style. For explicating the relationship between OWL DL and DLs, in Table 1 we report some OWL DL specific constructs, which can be mapped into DL expressions (see, e.g. [42]).

### 2.2 Disjunctive logic programs

#### 2.2.1 Syntax.

We follow [30, 56]. Consider an arbitrary first order language that contains infinitely many variable symbols, finitely many constants and function symbols and predicate symbols. We denote variables by lower case letters from the end of the alphabet, constants by lowercase letters from the beginning of the alphabet and predicates with capitals (as usual, metavariables may have a subscript or a superscript).

However, often in logic programming function symbols are not considered for computational reasons.
A term \( t \) is inductively defined as either a constant, a variable or an expression of the form \( f(t_1, \ldots, t_n) \), where all \( t_i \) are terms and \( f \) is a \( n \)-ary function symbol.

An atom is of the form \( P(t_1, \ldots, t_n) \), where all \( t_i \) are terms and \( P \) is a \( n \)-ary predicate symbol. Ground atoms are atoms without variables. A literal \( L \) is either a positive literal \( L = A \), or a negative literal \( L = \neg A \), where \( A \) is an atom. A ground literal is a literal without variables. An extended literal is a literal or an expression of the form \( \text{not}(L) \) ("\( L \) is not provable"), where \( L \) is a literal. A ground extended literal is an extended literal without variables. For a set \( X \) of extended literals, \( X^- = \{ \text{not}(L) : \text{not}(L) \in X \} \), while \( \neg X = \{ \neg L : L \in X; \text{\( L \) literal} \} \), where we define \( \neg \neg A = A \).

A disjunctive logic program \( \mathcal{P} \), is a finite set of rules of the form

\[
\gamma \leftarrow \delta,
\]

where \( \gamma \) and \( \delta \) are finite sets of extended literals. For ease, we may omit the graph brackets in a rule. \( \gamma \) is called the head of the rules, while \( \delta \) is called the body. The head is interpreted as the disjunction of its components, while the body is interpreted as a conjunction. A fact is a rule with empty body, while a constraint is a rule with empty head.

We call programs where for each rule \( \gamma^- \cup \delta^- = \emptyset \), programs without negation as failure (naf). Programs, where in each rule \( |\gamma| = 1 \) are called normal. Programs without naf, containing positive literals only, are called positive. Programs that do not contain variables are called ground.

2.2.2 Semantics.

For a program \( \mathcal{P} \), and a (possibly infinite) non-empty set of terms \( H \), such that every term which may be constructed from the constants an function symbols appearing in \( \mathcal{P} \), is in \( H \), we call \( \mathcal{P}_H \) the grounded program obtained from \( \mathcal{P} \) by substituting every variable in \( \mathcal{P} \) by every possible term in \( H \). Note that \( \mathcal{P}_H \) may contain an infinite number of rules (if \( H \) is infinite). The universe of a grounded program \( \mathcal{P} \) is the (possibly infinite) non-empty set of terms \( H_\mathcal{P} \) appearing in \( \mathcal{P} \). Note that \( H_{\mathcal{P}_H} = H \).

The base of a grounded program \( \mathcal{P} \) is the (possibly infinite) set \( B_\mathcal{P} \) of ground atoms that can be constructed using the predicate symbols in \( \mathcal{P} \) with the terms in \( H_\mathcal{P} \).

An interpretation \( I \) of a grounded program \( \mathcal{P} \) is any consistent set of literals being a subset of \( B_\mathcal{P} \cup \neg B_\mathcal{P} \). \( I \) satisfies a literal \( L \) iff \( L \in I \). Furthermore, we say that \( I \) satisfies an extended literal \( \text{not}(L) \) iff \( I \) does not satisfy \( L \), i.e. \( L \notin I \).

An interpretation \( I \) of a grounded program \( \mathcal{P} \) without naf satisfies a rule \( \gamma \leftarrow \delta \) iff \( \gamma \cap I \neq \emptyset \) whenever \( \delta \subseteq I \). An interpretation \( I \) is a model of program \( \mathcal{P} \) without naf if it satisfies every rule in \( \mathcal{P} \); \( I \) is a minimal model of \( \mathcal{P} \) if \( I \) is a model of \( \mathcal{P} \) and there is no model \( J \subset I \) of \( \mathcal{P} \).

For a grounded program \( \mathcal{P} \) and an interpretation \( I \), the Gelfond-Lifschitz transformation [30, 56], is the program \( \mathcal{P}^I \) without naf, obtained by deleting in \( \mathcal{P} \),

1. each rule that has \( \text{not}(L) \) in its body with \( L \in I \);
2. each rule that has \( \text{not}(L) \) in its head with \( L \notin I \); and
3. all \( \text{not}(L) \) in the bodies and heads of the remaining rules.

Finally, an interpretation of a program \( \mathcal{P} \) (possibly not grounded) is a pair \( I = (H, I) \), such that \( I \) is an interpretation of the grounded program \( \mathcal{P}_H \). An interpretation \( I = (H, I) \) of a program \( \mathcal{P} \) is a stable model of \( \mathcal{P} \) iff \( I \) is a minimal model of \( \mathcal{P}_H \). It
can easily be shown that, if $I = (H, I)$ is a stable model of $\mathcal{P}$, then $I$ is a model of $\mathcal{P}_H$.

We say that a program $\mathcal{P}$ entails a ground extended literal $L$, denoted $\mathcal{P} \models L$ iff every stable model of $\mathcal{P}$ satisfies $L$. Note that $\mathcal{P} \models \text{not}(L)$, where $L$ is a literal, if every stable model satisfying $\mathcal{P}$ does not satisfy $L$.

Notice that the universe $H$ is not fixed to be the Herbrand universe of the terms which can be build from the constants and function symbols appearing in $\mathcal{P}$.

3 Description Logic Programs

In this section we introduce Description Logic Programs (DLPs), which are a novel combination of description logic terminologies with disjunctive logic programs.

3.1 Syntax.

A Description Logic Program (DLP) is a pair $\mathcal{D}P = \langle T, \mathcal{P}\rangle$, where $T$ is a terminology and $\mathcal{P}$ is a disjunctive logic program. Role names and concept names appearing in $T$ may appear in the body as well as in the head of a rule $\gamma \leftarrow \delta \in \mathcal{P}$ and are managed as unary and binary predicates, respectively. Note that usually, concept and role names that appear in $T$ are not allowed to appear in the head of the rules (except for representing facts) 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. We do not impose this syntactical restriction, as from a semantics point of view it is unproblematic.

3.2 Semantics.

An interpretation for $\mathcal{D}P = \langle T, \mathcal{P}\rangle$ is a pair $I = (H, I)$, where $I$ is an interpretation for $\mathcal{P}$ and $(\Delta^T, \cdot^T)$ is an interpretation for $T$, where $\Delta^T = H$, and for concept names $A$ and roles names $R$, $A^T = \{ t \mid A(t) \in I \}$ and $R^T = \{ (t, t') \mid R(t, t') \in I \}$, respectively.

An interpretation $I = (H, I)$ is a model of $\mathcal{D}P = \langle T, \mathcal{P}\rangle$ iff

1. $(\Delta^T, \cdot^T)$ satisfies $T$;
2. $I$ satisfies $\mathcal{P}_H^I$;
3. $I$ is a as small as possible.

The definition of entailment is as usual.

Example 1 Consider $\mathcal{D}P = \langle T, \mathcal{P}\rangle$, with

$$
T = \{ 
A = \forall R. \neg C \\
B = \forall R. D \\
E = C \cap D 
\}
$$

$$
\mathcal{P} = \{ 
P(x) \leftarrow A(x) \\
P(x) \leftarrow R(x, y), E(y), \text{not}(Q(y)) \\
B(a) \leftarrow \}
$$

Note that $\mathcal{P}_H$ contains grounded extended literals.
Then, any model $I = (H, I)$ of $\mathcal{DP}$ is such that $a \in H$, $B(a) \in I$, $Q(a) \not\in I$ and either $A(a) \in I$ or $A(a) \not\in I$, i.e. $a^2 \in A^2$ or $a^2 \not\in A^2$. In the latter case, $a^2 \not\in (\forall R. \neg C)^2$, i.e. $a^2 \in (\exists R.C)^2$ holds. As $a^2 \in B^2$, it follows that $a^2 \in (\exists R.(C \cap D))^2$ and, thus, $a^2 \in (\exists R.E)^2$. As a consequence, any model of $\mathcal{DP}$ is one-to-one to $I_1 = (H, I_1)$ with universe $H$ containing $a$ and $I_1 = \{B(a), A(a), P(a)\}$, while $I_2 = \{B(a), R(a, b), C(b), D(b), E(b), P(a)\}$, for some $b \in H$. Indeed, in case of $I_1$, we have that $\mathcal{P}^I_{H}$ is

$$\mathcal{P}^I_{H} = \{ P(a) \leftarrow A(a) \} .$$

The it can be verified that

1. $(\Delta^1_1, \Delta^2_1)$ satisfies $T$;
2. $I_1$ satisfies $\mathcal{P}^I_{H}$;
3. $I_1$ is as small as possible.

Notice that indeed $I_1$ is as small as possible. In fact, the counter example candidate $I' = \{B(a)\} \subseteq I_1$ is in effect a model of $\mathcal{P}^I_{H}$, but then $(\Delta^1_1, \Delta^2_1)$ does not satisfy $T$. The argument for $I_2$ is similar. Therefore, it follows that $\mathcal{DP} \models P(a)$.

Note that $\mathcal{DP} \not\models R(a, a)$, as the universe $H$ is not fixed to be the Herbrand universe of $\mathcal{P}$. In case $H$ is requested to be the Herbrand universe of $\mathcal{P}$ then $\mathcal{DP} \models R(a, a)$.

The following example shows a less abstract case based on the same principle of Example 1.

**Example 2** Consider the following database schema of a company database \footnote{This example is taken from http://www.inf.unibz.it/~franconi/dl/course/}. let $\mathcal{DP} = (T, \mathcal{P})$ be such that

$T = \{ \text{Employee} \sqsubseteq (\forall \text{OfficeMate.Employee}) \cap (\forall \text{SuperVised.Manager}) \\
\quad \text{Manager} \sqsubseteq \text{Employee} \\
\quad \text{Manager} = \text{AreaManager} \sqcup \text{TopManager} \\
\quad \text{AreaManager} = \text{TopManager} \sqcup \text{\bot} \}$

$\mathcal{P} = \{ Q(x) \leftarrow \text{SuperVised}(x, y), \text{TopManager}(y), \text{OfficeMate}(y, z), \text{AreaManager}(z) \\
\quad \text{SuperVised}(\text{john}, \text{kim}) \leftarrow \\
\quad \text{SuperVised}(\text{john}, \text{mary}) \leftarrow \\
\quad \text{OfficeMate}(\text{mary}, \text{kim}) \leftarrow \\
\quad \text{OfficeMate}(\text{kim}, \text{paul}) \leftarrow \\
\quad \text{Manager}(\text{kim}) \leftarrow \\
\quad \text{TopManager}(\text{mary}) \leftarrow \\
\quad \text{AreaManager}(\text{paul}) \leftarrow \} .$

$Q(x)$ acts as our query. It turns out that $\mathcal{DP} \models Q(\text{john})$, i.e. $\text{john}$ is an answer to our query. Indeed, similarly as in Example 1, $\text{john}$ is either a $\text{TopManager}$ or an $\text{AreaManager}$. In the former case $\text{john}$ is supervised by the $\text{TopManager} \text{ jim}$, which has the $\text{AreaManager paul as OfficeMate}$. In the latter case $\text{john}$ is supervised by the $\text{TopManager mary}$, which has the $\text{AreaManager jim as OfficeMate}$. 

The following example shows an ontology integration problem as well as the problem of distributed search among different resources.

**Example 3** Informally, an ontology integration system may be seen as a triple \( \langle G, S, M \rangle \) ([54]), where \( G \) is the global ontology, while \( S \) are the source ontologies and \( M \) is mapping between \( S \) and \( G \). Under this view, let us consider the following scenario. Let us assume that there are two resources \( R_1 \) and \( R_2 \) in \( S \) based on the following ontologies.

Let us assume that the objects in the resources are documents, which may semantically be annotated with terms of the ontologies (see, e.g. [61] for a formal framework), taken from the Universal Decimal Classification (UDC) system.\(^7\) Essentially, (i) \( R_1 \) is about movies is general; and (ii) \( R_2 \) is about movies’ comments.

\[
T_1 = \{ \text{Movie} \sqsubseteq (\exists \text{MovieID} . \text{Identifier}) \sqcap \exists \text{MovieTitle} . \text{AnyText} \sqcap \exists \text{MovieComment} . \text{AnyText} \}
\]

\[
T_2 = \{ \text{MovieReview} \sqsubseteq (\exists \text{MovieID} . \text{Identifier}) \sqcap \exists \text{MovieTitle} . \text{TextDescription} \sqcap \exists \text{HasCritique} . \text{TextDescription} \}
\]

Let us assume that a search agent is aware of the two resources and has its own ontology (which acts as the global ontology \( G \)):

\[
T = \{ \text{Movie} \sqsubseteq (\exists \text{HasCode} . \text{MovieCode}) \sqcap \exists \text{HasTitle} . \text{TextDescription} \sqcap \exists \text{Year} . \text{Number} \sqcap \exists \text{About} . \text{TextDescription} \sqcap \exists \text{HasDirector} . \text{MovieDirector} \\
\text{EuropeanDirector} \sqsubseteq \text{MovieDirector} \sqcap \forall \text{Born} . \text{EuropeanCountry} \\
\text{MovieReview} \sqsubseteq (\exists \text{HasCode} . \text{MovieCode}) \sqcap \exists \text{HasTitle} . \text{TextDescription} \sqcap \exists \text{HasCritique} . \text{TextDescription} \}
\]

We ask the search agent to search for title and critique of movies in 2003, conducted by European directors. Such a request may be written by means of the rule

\[
Q(t, r) \leftarrow \text{Movie}(x), \text{HasTitle}(x, t), \text{Year}(x, 2003), \text{HasDirector}(x, y), \text{EuropeanDirector}(y), \text{MovieReview}(x), \text{HasCritique}(x, r)
\]

\(^7\)UDC is the world’s foremost multilingual classification scheme for all fields of knowledge. See, http://www.udcc.org/.
In order to satisfy the request, the search agent should be able to determine a link between its own terminology and the one of the resources he is aware of. For instance, suppose we were able to determine a mapping (manually, automatically, semi-automatically [72]) between then agents ontology and the resources’ ontologies and that it looks like as follows:

1. the mapping to resource $R_1$ contains the rules

   \[
   \begin{align*}
   \text{Movie}(x) & \leftarrow \text{M}(x) \\
   \text{HasTitle}(x, y) & \leftarrow \text{MTitle}(x, y) \\
   \text{Year}(x, y) & \leftarrow \text{MYear}(x, y) \\
   \text{HasDirector}(x, y) & \leftarrow \text{MDirector}(x, y) \\
   \text{About}(x, y) & \leftarrow \text{MSummary}(x, y) \\
   \text{MovieReview}(x) & \leftarrow \text{M}(x) \\
   \text{HasCritique}(x, y) & \leftarrow \text{MSummary}(x, y) 
   \end{align*}
   \]

2. the mapping to resource $R_2$ contains the rules

   \[
   \begin{align*}
   \text{Movie}(x) & \leftarrow \text{MC}(x) \\
   \text{HasCode}(x, y) & \leftarrow \text{MovieID}(x, y) \\
   \text{HasTitle}(x, y) & \leftarrow \text{MovieTitle}(x, y) \\
   \text{About}(x, y) & \leftarrow \text{MovieComment}(x, y) \\
   \text{MovieReview}(x) & \leftarrow \text{MC}(x) \\
   \text{HasCritique}(x, y) & \leftarrow \text{MovieComment}(x, y) .
   \end{align*}
   \]

It is easily verified that the above mapping suffices to solve the query in terms of calls to the resources $R_i$.

Less obvious in the above example, is how to solve queries asking a search agent, for instance, to find movies conducted by European directors “about the difficulties of life” or the critiques addressing the “social impact of a movie”. It is evident that any relevant answer (a retrieved object) provided by a search agent to such requests is not a matter of just “crisp” yes or no decisions (a document satisfies a request or it does not satisfy it), but rather the answer is accompanied with a degree of certainty (the so-called Retrieval Status Value (RSV), in Information Retrieval terms [77, 5]), which is the agent’s degree of believing an object to satisfy the request, and the list of answers is ordered according to their RSV. This is exactly what happens in current Web search engines and we do not expect that this will change either in case Web objects are semantically annotated as well. Furthermore, as described in the introduction, an agent must automatically know (i) where to search; (ii) how to query different resources; and (iii) how to combine information from diverse resources. As information resources continue to proliferate, these problems of resource selection, schema mapping and data fusion become major obstacles to information access for which accurate automated tools are desired (they are an ineffective manual task). Any successful solution to distributed search (of documents and services) in the Semantic
Web, like in Information Retrieval, should envisage a fully automatic process in the large scale. Towards this end, we need a suitable representation language and likely, it should be able to deal with the management of the inherent uncertainty in the above tasks. A preliminary language proposal towards this direction is the subject of the next section.

4 Description Logic Programs with Uncertainty

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 the real unit interval \([0, 1]\). The associated value \(c \in [0, 1]\) indicates to which extent a predicate is true.

4.1 Syntax.

From a syntax point of view, terminological axioms remain unchanged, i.e. of the form \(C \sqsubseteq D\) (resp. \(R \sqsubseteq R'\)) or of the form \(C = D\) (resp. \(R = R'\)). Unlike classical DLs, we will be able to specify to which extent \(c \in [0, 1]\) (how certain it is that) a constant \(a\) is an instance of the concept \(C\).

For rules, we have major modifications, which are inspired on the so-called Generalized Annotated Logic Programs (GAP) framework of Kifer and Subrahmanian [50]. So, let us define an annotation function of arity \(n\) to be a total and computable function \(f : ([0, 1])^n \rightarrow [0, 1]\). Assume a new alphabet of annotation variables, which will denote a value in \([0, 1]\) and can only appear in so-called annotation terms. An annotation item, \(\kappa\), is defined inductively: (i) as a real \(c \in [0, 1]\), or as an annotation variable \(\nu\), or (ii) is of the form \(f(\kappa_1, \ldots, \kappa_n)\), where \(f\) is an \(n\)-ary annotation function and all \(\kappa_i\) are annotation items. An annotation term, \(\lambda\), is of the form \([\kappa, \kappa']\), where \(\kappa\) and \(\kappa'\) are annotation items. Annotation terms are supposed to denote subintervals of \([0, 1]\). Now, let \(L\) be a literal and \(\lambda\) an annotation term.

A \mu literal, denoted \(\mu L\), is of the form \(L : \lambda\). The intended meaning is that “the certainty of \(L\) lies in the interval \(\lambda\)”. An extended \mu literal is of the form \(\text{not(}\mu L\text{)}\), where \(\mu L\) is a \mu literal. The intended meaning of \(\text{not(}L : \lambda\text{)}\) is that “it is not provable that the certainty of \(L\) lies in the interval \(\lambda\)”. For instance, following [61, 82], an image \(i\) may be annotated with the \mu literal \(\text{About(}i, \text{landscape)} : \lambda\), indicating the extent to which an automated image annotation (classification) tool (see, e.g. [31, 93]) is certain that the image \(i\) is about a landscape.

A \mu disjunctive logic program (or, simply \mu program), \(\mu P\), is a finite set of \mu rules of the form \(\gamma \leftarrow \delta\), where \(\gamma\) and \(\delta\) are finite sets of extended \mu literals. Finally, a \mu Description Logic Program (\muDLP), denoted \(\mu\text{DLP}\), is a pair \((T, \mu P)\), where \(T\) is a terminology and \(\mu P\) is a \mu program.

Example 4 Consider Example 3. Let us assume that the resource \(R_1\) annotates the movies with Dublin Core metadata records. Roughly a Dublin Core metadata record contains 15 attributes describing a document and consists of the attributes title, creator, subject, description, publisher, contributor, date, type, format, identifier, source, ...
language, relation, coverage and rights. We may thus generate a schema mapping rule relating the term About of the agent’s ontology to \( R_1 \) as follows:

\[
\begin{align*}
\text{Map(about.title)}: [0.6, 1] & \quad \leftarrow \\
\text{Map(about.subject)}: [0.7, 1] & \quad \leftarrow \\
\text{Map(about.description)}: [0.9, 1] & \quad \leftarrow
\end{align*}
\]

About\((x, y): [\nu, 1]\) \quad \leftarrow \quad \text{Map(about.title)}: [\nu, 1], \text{Title}(x, y): [\nu', 1]

About\((x, y): [\nu, 1]\) \quad \leftarrow \quad \text{Map(about.subject)}: [\nu, 1], \text{Subject}(x, y): [\nu', 1]

About\((x, y): [\nu, 1]\) \quad \leftarrow \quad \text{Map(about.description)}: [\nu, 1], \text{Description}(x, y): [\nu', 1]

The first three facts represent the relationships (with their degree of certainty) an automated schema mapping tool was able to discover among the property about and the properties title, subject and description. The second set of rules uses these discovered relationships to establish the mapping among the agent’s ontology and the one of the resource \( R_1 \). Furthermore, keyword search may be supported by using rules like

\[
\begin{align*}
\text{SearchMovieAbout}(x, y): [\nu \cdot \nu', 1] & \quad \leftarrow \quad \text{About}(x, y'): [\nu, 1], \text{Sim}(y, y'): [\nu', 1]
\end{align*}
\]

In the above rule, the variable \( x \) denotes a movie, \( y \) denotes the text to be searched for, \( y' \) denotes the text describing the content of the movie \( x \), while \( \text{Sim}(y, y') \) is a built-in predicate computing the similarity (using usual text similarity measures [5]) among the text denoted by \( y \) and \( y' \) and \( \nu' \) is the computed RSV. Note that such a rule takes into account both the uncertainty of the schema mapping as well as the uncertainty of the “aboutness”.

4.2 Semantics.

We proceed as follows. We specify the semantics of the terminological component first and then the one of the rule component and conclude with defining the models of a \( \mu \)DLP.

The interpretation of concepts and roles and the semantics of terminological axioms is as in [83]. Formally, a \( \mu \)-interpretation is a pair \( I = (\Delta_I, \cdot_I) \), where \( \Delta_I \) is the domain and \( \cdot_I \) is an interpretation function mapping

- a concept \( C \) into a function \( C^\mathcal{I}: \Delta_I \rightarrow [0, 1] \); and
- a role \( R \) into a function \( R^\mathcal{I}: \Delta_I \times \Delta_I \rightarrow [0, 1] \).

If \( C \) is a concept then \( C^\mathcal{I} \) will naturally be interpreted as the membership degree function of the concept \( C \) w.r.t. \( I \), i.e. if \( d \in \Delta_I \) is an object of the domain \( \Delta_I \) then \( C^\mathcal{I}(d) \) gives us the certainty degree of being the object \( d \) an instance of the concept \( C \) under the \( \mu \)-interpretation \( I \). Similarly for roles.

The interpretation function \( \cdot^\mathcal{I} \) has also to satisfy the following equations: for all \( d \in \Delta_I \),
\[
\begin{align*}
\tau^T(d) & = 1 \\
\lambda^T(d) & = 0 \\
(C \cap D)^T(d) & = \min(C^T(d), D^T(d)) \\
(C \cup D)^T(d) & = \max(C^T(d), D^T(d)) \\
(-C)^T(d) & = 1 - C^T(d) \\
(\forall R.C)^T(d) & = \inf_{d' \in \Delta^T} \{\max(1 - R^T(d, d'), C^T(d'))\} \\
(\exists R.C)^T(d) & = \sup_{d' \in \Delta^T} \{\min(R^T(d, d'), C^T(d'))\}.
\end{align*}
\]

These equations are the standard interpretation of conjunction, disjunction, negation and quantification, respectively.

Concerning terminological axioms, a \(\mu\)interpretation \(I\) satisfies \(C \subseteq D\) iff for all \(d \in \Delta^T, C^T(d) \leq D^T(d)\). Similarly, \(\mu\)interpretation \(I\) satisfies \(R \subseteq R'\) iff for all \(\{d, d'\} \subseteq \Delta^T, R^T(d, d') < R'^T(d, d')\). Finally, \(\mu\)interpretation \(I\) satisfies (is a model of) a terminology \(T\) iff \(I\) satisfies each element in it, which concludes the semantics for terminological components.

Concerning \(\mu\)programs we have the following definitions. In order to avoid straight-forward repetition, if not stated otherwise, definitions related to \(\mu\)disjunctive logic programs, parallels those for disjunctive logic programs. Furthermore, in grounding a \(\mu\)literal \(L: \lambda\), we assume that the annotation term \(\lambda\) is grounded as well, i.e. annotation variables are replaced with values in \([0, 1]\) and annotation items of the form \(f(\kappa_1, \ldots, \kappa_n)\) are replaced with the result of the computation of \(f(\kappa_1, \ldots, \kappa_n)\). Note that a grounded \(\mu\)program \(\mu P\) may contain an infinite number of rules due to the grounding of annotation terms. For a grounded \(\mu\)program \(\mu P\), \(B_{\mu P}\) is the set of ground atoms \(A\) that can be constructed using the predicate symbols in \(\mu P\), where \(A\) is grounded with the terms in \(H_{\mu P}\) (annotations terms are not considered).

An \(\mu\)interpretation \(I\) of a grounded \(\mu\)program \(\mu P\) is any (possibly partial) function \(I: B_{\mu P} \rightarrow [0, 1]\) (some ground atoms may be left unspecified). The set of defined atoms in \(I\) is denoted \(\text{def}(I)\). In the following, whenever we write \(I(A)\), we assume that \(A \in \text{def}(I)\). We extend \(I\) to literals \(L = \neg A\) in the obvious way: \(I(L) = 1 - I(A)\). A \(\mu\)interpretation \(I\) satisfies a ground \(\mu\)literal \(L: \lambda\) iff \(I(L) \in \lambda\). Note that we can always assume that \(\mu\)literals are positive, by replacing \(\neg A: [\kappa_1, \kappa_2]\) with \(A: [-\kappa_2, -\kappa_1]\). Like for disjunctive logic programs, we say that \(I\) satisfies an extended \(\mu\)literal \(n\text{ot}(L: \lambda)\) iff \(I\) does not satisfy \(L: \lambda\), i.e. \(I(L) \notin \lambda\).

An \(\mu\)interpretation \(I\) of a grounded program \(\mu P\) without \text{naf} satisfies a rule \(\gamma \leftarrow \delta\) iff if \(I\) satisfies every \(\mu\)literal in \(\delta\) then \(I\) satisfies some \(\mu\)literal in \(\gamma\). An \(\mu\)interpretation \(I\) is a \(\mu\)model of program \(\mu P\) without \text{naf} iff it satisfies every rule in \(\mu P\).

**Example 5** Consider the grounded \(\mu\)program \(\mu P\) without \text{naf}, with two facts

\[ A: [0.2, 0.7], B: [0.3, 0.6] \leftarrow \]
\[ C: [0.1, 0.3] \leftarrow \]

Let us consider the following two partial functions \(\mathcal{F}_1\) and \(\mathcal{F}_2\), assigning to atoms intervals and defined as follows:

\[
\begin{align*}
\mathcal{F}_1(A) & = [0.2, 0.7] \\
\mathcal{F}_1(C) & = [0.1, 0.3]
\end{align*}
\]

13
It is easily verified that for each \( \mu \) model \( I \) of \( \mu P \), we have that for some \( i = 1, 2 \), 
\[ \text{def}(I) \subseteq \text{def}(I_i) \] 
and for all ground atoms \( P \in \text{def}(I_i), \) \( I(P) \in \mathcal{I}(P) \). Essentially, the \( \mathcal{I}_i \) are minimal in terms of the atoms defined and the intervals are the ‘most precise’ intervals that can be inferred.

In the following we formally define the above concept of “interval interpretation”. Let \( C[0,1] \) be the set of all closed sub-intervals of \([0,1]\). We also add \( \emptyset \) to \( C[0,1] \), called the empty interval. We will use it to manage inconsistencies among intervals. For two intervals \( \sigma_1 \) and \( \sigma_2 \) in \( C[0,1] \), we define

\[
\sigma_1 \preceq \sigma_2 \text{ iff } \sigma_2 \subseteq \sigma_1 .
\]

Similarly, \( \sigma_1 \prec \sigma_2 \text{ iff } \sigma_2 \subset \sigma_1 \). Furthermore, we define \( \neg [c,c'] = [1-c',1-c] \). The \( \preceq \)-least interval is \([0,1]\), the \( \preceq \)-greatest interval is \( \emptyset \).

An interval interpretation \( \mathcal{I} \) of a grounded program \( \mu P \) without naf, is a (possibly partial) function \( \mathcal{I} : B_{\mu P} \rightarrow C[0,1] \). An interval interpretation \( \mathcal{I} \) is a representative of a whole family of \( \mu \) interpretations \( I \) : we write \( I \in \mathcal{I} \) iff \( \text{def}(I) = \text{def}(\mathcal{I}) \) and for all \( A \in \text{def}(\mathcal{I}), \) \( I(A) \in \mathcal{I}(A) \) (see Example 5). We extend interval interpretations \( \mathcal{I} \) to ground literals \( L = \neg A \) as usual: \( \mathcal{I}(L) = \neg \mathcal{I}(A) \).

For interval interpretations \( \mathcal{I}_1 \) and \( \mathcal{I}_2 \),

\[
\mathcal{I}_1 \preceq \mathcal{I}_2 \text{ iff } \text{def}(\mathcal{I}_1) \subseteq \text{def}(\mathcal{I}_2) \\
\text{and for all } A \in \text{def}(\mathcal{I}_1), \mathcal{I}_1(A) \preceq \mathcal{I}_2(A) .
\]

\( \mathcal{I}_1 \prec \mathcal{I}_2 \text{ iff } \mathcal{I}_1 \preceq \mathcal{I}_2 \) and either \( \text{def}(\mathcal{I}_1) \subset \text{def}(\mathcal{I}_2) \) or for some \( A \in \text{def}(\mathcal{I}_1), \mathcal{I}_1(A) \prec \mathcal{I}_2(A) \). The \( \preceq \)-greatest interval interpretation, \( \mathcal{I}_\top \), assigns to all ground atoms in \( B_{\mu P} \) the empty interval, while the \( \preceq \)-least interval interpretation, \( \mathcal{I}_\bot \), is undefined on all ground atoms in \( B_{\mu P} \), i.e. \( \text{def}(\mathcal{I}_\bot) = \emptyset \), meaning essentially that the certainty of all the atoms is unknown.

An interval interpretation \( \mathcal{I} \) satisfies a ground \( \mu \) literal \( \delta \lambda \) iff \( \lambda \preceq \mathcal{I} (L) \). An interval interpretation \( \mathcal{I} \) of a grounded program \( \mu P \) without naf satisfies a rule \( \gamma \leftarrow \delta \) iff if \( \mathcal{I} \) satisfies every \( \mu \) literal in \( \delta \) then \( \mathcal{I} \) satisfies some \( \mu \) literal in \( \gamma \). An interval interpretation \( \mathcal{I} \) is an interval model of program \( \mu P \) without naf if it satisfies every rule in \( \mu P \). Furthermore, \( \mathcal{I} \) is minimal as well iff there is no interval model \( \mathcal{I}' \prec \mathcal{I} \) of \( \mu P \). For instance, in Example 5, \( \mathcal{I}_1 \) and \( \mathcal{I}_2 \) are the only two minimal interval models of \( \mu P \).

Like in [50], it can easily be shown that if \( \mu P \) is normal grounded program without naf, then there is an unique minimal interval model for \( \mu P \), which is the \( \preceq \)-least fixed-point, \( f_p(T_{\mu P}) \), of the following \( T_{\mu P} \) monotone operator: for all \( A \in B_{\mu P} \),

\[
T_{\mu P}(\mathcal{I})(A) = \bigcap \{ \lambda \mid A; \lambda \leftarrow \delta \in \mu P, \mathcal{I} \text{ satisfies each } \mu \text{ literal in } \delta \} .
\]

Note that \( T_{\mu P} \) is not continuous. In fact the grounded \( \mu \) program without naf, containing all ground instances of the rules
A: [0, 1] \rightarrow \\
A: [\nu + \frac{1}{2}, \nu'] \rightarrow A: [\nu, \nu'] \\
B: [1, 1] \rightarrow A: [1, 1]

has unique minimal interval model \( \mathcal{J}(A) = \mathcal{J}(B) = [1, 1] \), which is obtained after \( \omega + 1 \) \( T_{gP} \) iterations over \( \mathcal{J}_L \), where \( \omega \) is the first limit ordinal. Note also that the least and unique interval model \( \mathcal{J} \) of the \( \mu \)program \( \{(A: [0, 0], (A: [1, 1])\}} \) assigns to \( A \) the empty set \( \emptyset \), which indicates that on \( A \) the \( \mu \)program is inconsistent and, thus, has no \( \mu \)model. We may use this property to manage inconsistencies of this kind, by allowing interpretations to map ground atoms into a new symbol, indicating that the \( \mu \)program is inconsistent on that atom. We will not investigate this issue further in this paper.

Given a grounded \( \mu \)program (possibly with naf) \( \mu P \) and an \( \mu \)interpretation \( I \) for \( \mu P \), the Gelfond-Lifschitz transformation, is the grounded positive \( \mu \)program \( \mu P^I \), obtained by deleting in \( \mu P \),

1. each rule that has not(\( \mu L \)) in its body and \( I \) satisfies \( \mu L \);
2. each rule that has not(\( \mu L \)) in its head and \( I \) does not satisfy \( \mu L \);
3. all not(\( \mu L \)) in the bodies and heads of the remaining rules.

Finally, an \( \mu \)interpretation of a \( \mu \)program \( P \) (possibly not grounded) is a pair \( I = (H, I) \), such that \( I \) is an \( \mu \)interpretation of the grounded program \( \mu P_H \). An \( \mu \)interpretation \( I = (H, I) \) of a \( \mu \)program \( \mu P \) is a stable \( \mu \)model of \( \mu P \) iff \( I \in \mathcal{J} \) for a minimal interval model \( \mathcal{J} \) of \( \mu P_H \). Finally, we say that a program \( \mu P \) entails a ground extended \( \mu \)literal \( \mu L \), denoted \( \mu P \models \mu L \) iff every stable \( \mu \)model of \( \mu P \) satisfies \( \mu L \). For instance, in Example 5, we have that any stable \( \mu \)model \( I \) of \( \mu P \) is such that \( I \in \mathcal{J}_1 \) or \( I \in \mathcal{J}_2 \).

**Example 6** Consider the following \( \mu \)programs \( \mu P_1, \mu P_2, \mu P_3 \) and \( \mu P_4 \), where

\[
\begin{align*}
\mu P_1 &= \{r_1, r_2\} , \quad \mu P_2 = \{r_1, r_3\} \\
\mu P_3 &= \{r_1, r_4\} , \quad \mu P_4 = \{r_1, r_5\}
\end{align*}
\]

and the rules \( r_i \) are

\[
\begin{align*}
r_1 : A: [0.6, 0.8] \rightarrow \\
r_2 : B: [0.4, 0.5] \rightarrow \text{not}(A: [0.2, 0.3]) \\
r_3 : B: [0.4, 0.5] \rightarrow \text{not}(A: [0.2, 0.7]) \\
r_4 : B: [\nu, \nu'] \rightarrow A: [\nu, \nu'] \\
r_5 : B: [\nu, \nu'] \rightarrow \text{not}(A: [\nu, \nu']).
\end{align*}
\]

It can be verified that for any stable \( \mu \)model \( I \) of \( \mu P_1 \) we have that

1. for \( \mu P_1 \), \( I(A) \in [0.6, 0.8] \) and \( I(B) \in [0.4, 0.5] \). Therefore, \( \mu P_1 \models B: [0.4, 0.5] \); 2. for \( \mu P_2 \), \( I(A) \in [0.6, 0.8] \) and if \( I(A) \in (0.7, 0.8) \) then \( I(B) \in [0.4, 0.5] \) else \( I \) is undefined on \( B \). Therefore, \( \mu P_2 \models [c, c'] \) for any \( c, c' \in [0, 1] \). But, \( \mu P_2 \models \text{not}(B: [c, c']) \) if \( [c, c'] \cap [0.4, 0.5] = \emptyset \); 3. for \( \mu P_3 \), \( I(A) \in [0.6, 0.8] \) and \( I(B) \in [0.6, 0.8] \). Therefore, \( \mu P_3 \models B: [0.6, 0.8] \).
We are ready now to define the semantics of $\mu$-DLPs. An interpretation for a $\mu$-DLP $\mu\mathcal{DP}$ is a pair $I = (H, I)$, where $I$ is a $\mu$-interpretation for the $\mu$-program $\mu\mathcal{P}$ and $(\Delta^I, \Xi^I)$ is a $\mu$-interpretation for the terminology $\mathcal{T}$, where $\Delta^I = H$, and for concept names $A$ and roles names $R$, $A^I(t) = I(A(t))$ and $R^I(t) = I(R(t, t'))$ $(t, t' \in H)$, respectively.

An interpretation $I = (H, I)$ for a $\mu$-DLP $\mu\mathcal{DP} = (\mathcal{T}, \mu\mathcal{P})$ is a $\mu$-model of $\mu\mathcal{DP}$ iff
1. $(\Delta^I, \Xi^I)$ is a model of the terminology $\mathcal{T}$;
2. $I \in \mathcal{I}$ and $\mathcal{I}$ is an interval model $\mathcal{I}$ of $\mu\mathcal{P}_H$;
3. $\mathcal{I}$ is as small as possible.

Entailment is defined as usual.

**Example 7** Let us consider the DLP of Example 1, but extended with uncertainty as follows. $\mu\mathcal{DP} = (\mathcal{T}, \mu\mathcal{P})$, where

\[
\mathcal{T} = \{ 
A = \forall R.R.C \\
B = \forall R.D \\
E = C \cap D 
\}
\]

\[
\mu\mathcal{P} = \{ 
B(a); [0.7, 1] \leftarrow \\
P(x); [\nu, \nu'] \leftarrow A(x); [\nu, \nu'] \\
P(x); [\nu, \nu'] \leftarrow R(x, y); [\nu, \nu'], E(y); [\nu, \nu'], not(R(y); [0.1, 0.2]) \}
\]

Consider the atom $P(a)$. Then, by reasoning by cases like in Example 1, it follows that $\mu\mathcal{DP} = P(a); [0.5, 1]$. In fact, the $\mu$-models $I = (H, I)$ of $\mu\mathcal{DP}$ are such that $a \in H$ and either $def(I) = \{B(a), A(b), P(a)\}$ or $def(I) = \{B(a), R(a, b), C(b), D(b), E(b), P(b)\}$, for some $b \in H$. Furthermore, in either case we have $I(P(a)) \geq \max(c, \min(0.7, 1 - c))$, for any $c \in [0, 1]$. As a consequence, for any $\mu$-model $I = (H, I)$ of $\mu\mathcal{DP}$, we have $I(P(a)) \geq 0.5$.

5 Reasoning support

5.1 Description logic programs

Reasoning in full DLPs is a difficult task, even for a small fragments of it\(^{10}\). In fact, from the results in [55], in which DLs are combined with function-free Horn rules, from [55, Theorem 5.1] it follows easily that the entailment problem in DLPs is undecidable. For instance, it suffices to allow "acyclic" terminologies with concept definitions only, i.e. of the form $A = C$ and there is no concept defined in terms of

\(^{10}\)See for instance, [4] as a source of complexity results for DLs, [17, 23] as a source of complexity results for LPs.
it self, the ∃R.C constructor and recursive function-free Horn rules, i.e. rules without naf, to guarantee undecidability.

In case we restrict our attention to DLPs in which Horn rules are considered only, we may easily translate DLPs into First-Order Logic (FOL) and use FOL theorem provers, like Vampire [76] to reason in the resulting first order theory. The use of FOL ensures that no restriction on the form of Horn rules and on the terminology is required and that all inferences were sound. In case decidability is desired, we have to restrict the form or the expressiveness of DLPs. [33, 55] provide many concrete suggestions in the restriction of the form of terminological axioms as well as of the form of rules to fit either within a tractable or decidable fragment of DLPs. On the other hand, [25] suggests to use a weaker semantics to be used to get decidable reasoning (see the section about related work).

In case of DLPs with naf, [25] shows concretely how to combine normal logic programs with naf with DLs, by weakening the semantics. In particular, concept and role predicates appearing in rules can be managed as system calls to a DL reasoner (in particular, RACER 11). On the other hand, if we rely on the semantics given in this paper, of course the entailment problem is undecidable as well. Towards the support of decidable entailment, e.g. [84] shows that under certain reasonable circumstances we may reduce the entailment problem in DLPs into an entailment problem for disjunctive logic programs. This has the considerable practical advantage that deciding an entailment problem in DLPs may be deferred to a disjunctive logic program reasoner like DLV [19] or smodels [68].

5.2 Description logic programs with uncertainty

Reasoning in µDLP is further complicated by the introduction of annotation terms. For instance, the simple example showing the non-continuity of the $T_{µP}$ immediate consequence operator is indicating an undecidability result for normal µprograms without naf, while their non-annotated counterpart is decidable (see, e.g. [24]). As a consequence, similarly as for DLPs, we have either to restrict the form or the expressiveness of µDLPs. An interesting source for suggestions, but applicable to normal rules without naf only, can be found in [50]. In the case of µDLPs with naf, another approach is shown in [84], where in particular it is shown that under certain restricted circumstances we may reduce the entailment problem in µDLPs into an entailment problem for disjunctive logic programs and, thus, state of the art disjunctive logic program reasoner may be applied. Of course, further investigations are required in this field.

6 Related work

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

Concerning µDLPs, the terminological component mainly relies on the work of Straccia [83], but it allows only a restricted form of terminological axioms. Concerning the rule component, as anticipated, it is inspired by the Generalized Annotated Logic Programming framework of Kifer and Subrahmanian [50], but extends it in two

11http://www.cs.concordia.ca/~haarslev/racer/
directions: disjunctions may appear in the head of rules; negation-as-failure and explicitly negated atoms may appear in rules. The semantics we devised in that case closely relates the one of [65], but there a probabilistic setting has been considered.

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 [20, 55, 43]. In particular, [20] 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 [55], which combines Horn rules with the description logic $\mathcal{ALCNR}$, where concepts and roles may appear in the body of Horn rules. Like in [20], [55] devices an SLD resolution integrated with an inference method for $\mathcal{ALCNR}$. Unfortunately, this approach is not easy applicable to full DLPs as negation-as-failure is present and rules heads are disjunctions. However, it may be applied to special cases, for which top-down resolution methods for the logic programming component are known (see, e.g. [14, 15, 73, 95]). Finally, [43] is similar to [55], 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 [91, 1, 87, 33, 36, 63]. We remark that (i) [91] reduces knowledge bases in the DL $\mathcal{ALC\Box}$ [10] ($\mathcal{ALC}$ with number restrictions) to open logic programs; (i) [1, 87] reduce reasoning in the DL $\mathcal{ALC\Box}$ [89, 88] ($\mathcal{ALC}$ with qualified number restrictions and inverse roles) to query answering from answer sets of normal logic programs; (iii) [33] shows how a subset of the DL $\mathcal{SHOIQ}$ [40, 39, 44] can be reduced to a subset of Horn programs (positive normal programs); (iv) [36] reduces the DL $\mathcal{SHIF}$ with transitive role closure to disjunctive logic programs (in [84] we use a similar reduction); and [63] 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 [2, 25, 98, 99]. Roughly, [25] combines the DL $\mathcal{SHOIN}(D)$, which is $\mathcal{SHOIN}$ with concrete domains (e.g. strings, integers, reals), with normal logic programs. The main characteristics of [25] lies in the use of a weaker semantics to be used to get decidable reasoning (but, e.g. it does not answer positively to the query of Example 2). Indeed, concept and role predicates appearing in rules can be managed as system calls to a DL reasoner. [2] proposes defeasible reasoning in normal logic programs combined with a reduction of a subset of OWL Lite to normal logic programs in similar manner as [33]. Finally, [98] (and similarly in [99]) is based on F-logic [48, 49].
7 Conclusions

Towards the integration of rules and ontologies in the Semantic Web, we have proposed a framework in order to combine logic programming under the answer set semantics with description logics, which stand behind ontology languages like DAML+OIL, OWL Lite and OWL DL. We have defined the new family of Description Logic Programs (DLPs), which combine DLs with Disjunctive Logic Programs under the answer set semantics: concepts, roles, explicit negation and negation as failure may appear both in the head as well as in the body of rules. We defined their abstract syntax and semantics and discussed related computational issues. While the combination of DLs with positive logic programs (Horn logics) is not new, the integration of DLs with disjunctive logic programs has not been addressed yet. We hope that DLPs may become the reference language of the combination of the two representation paradigm to which alternative formalizations may be compared.

We further have shown that in the realistic setting of distributed search in the Semantic Web, where an agent is asked for searching relevant objects from the numerous information resources he has access to, there is an implicit requirement of uncertainty management at various levels: document content representation (annotation), document retrieval, automated resource selection, automated schema mapping and retrieval results merging. We have proposed a framework to extend DLPs towards this direction by proposing to augment DLPs with annotation terms indicating to which degree a literal is certain. As for DLPs, we defined their abstract syntax, semantics and discussed some computational issues.

The main direction for future work involves the computational aspect. Currently, there are three alternatives worth to be investigated to provide reasoning support to (subsets of) DLPs: (i) translating DLPs into disjunctive logic programming [91, 1, 87, 33, 36, 63]; (ii) weakening the semantics such that ontology concepts and properties are managed like procedural attachments [25, 64]; or (iii) to rely on methods, which combine logic programming SLD refutation with DLs tableaux like refutation methods [20, 55].

While there is already substantial work from which the computational issues addressed by DLPs may take advantage of, little is known concerning µDLPs and, thus, this may be an appealing area of discovery.

It might well be of interest to consider different approaches towards the management of uncertainty in DLPs, by relying on alternative notions of uncertainty. For instance, related to DLs, we may mention

- **probability theory:** [32, 35, 46, 51, 80];
- **possibility theory:** [38];
- **fuzzy theory:** [16, 37, 83, 90, 100];
- **multi-valued theory:** [83, 85]

which have to be coupled with the corresponding rule component based on

- **probability theory:** [18, 29, 52, 58, 65, 66, 67, 97];
- **possibility theory:** [21];
- **fuzzy theory:** [13, 45, 81, 92, 94];
- **multi-valued theory:** [26, 27, 47, 50, 53, 57, 59, 60, 86].

19
References


22


