Uncertainty in Description Logic Programs

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

January 21, 2004

Abstract

We present a new family of representation languages, called Description Logic Programs (DLPs) and DLPs with uncertainty ($\mu$DLPs). The former combine the expressive power of description logics and disjunctive logic programs, while the latter are DLPs in which the management of uncertainty is based on so-called annotation terms, inspired by the generalized annotated logic programming framework [10].

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

1 Introduction

In the last decade a substantial amount of work has been carried out in the context of Description Logics (DLs) [2]. DLs provide a simple well-established Tarski-style declarative semantics to capture the meaning of the most popular features of structured representation of knowledge (see the DL community home page http://dl.kr.org/). Notably, for instance, DLs find a natural application in the context of the Semantic Web 1 [3], in order to define the ontology of an information source (informally, an ontology is a hierarchical description of important concepts in a particular domain, along with the description of the properties of the instances of each concept)–see, e.g. [9].

1www.semanticweb.org
Of course, each information source may have its own ontology. For instance, the picture above depicts a slice of the scenario of two Computer Science Departments. It turns out that, for instance, if we would like to integrate the ontologies, like required in heterogeneous data integration (see, e.g., [12], or making an agent to perform a search task among different information sources (which is known in the Information Retrieval literature as Distributed Information Retrieval, or Metasearch [1]) some rules mapping concepts from one ontology to the other may be required. Furthermore, we have to cope with the uncertainty inherent to those mappings. For instance, in the picture, the staff members of department A are members of the technical staff class of department B, with e.g. a certain probability.

Logic programs (LPs) and, more importantly, the management of uncertainty in LPs (see, e.g. [10, 11]) seem to be the ‘reference’ representation languages for these purposes. Indeed, numerous frameworks have been proposed over the last 20 years for the management of uncertainty in LPs. Essentially, they differ in the underlying notion of uncertainty (e.g. probability theory, fuzzy set theory, multi-valued logic, possibilistic logic) and how uncertainty values, associated to rules and facts, are managed.

Guided by our long term project on distributed search on the semantic web, the topic of this paper is to combine DLs, LPs and the management of uncertainty into an uniform framework. While the combination of DLs and LPs is not new (see, e.g. [5, 7, 8, 13]), their extension to the management of uncertainty, to the best of our knowledge, has not yet been investigated. In particular, we will integrate description logics with disjunctive logic programs with negation as failure (see, e.g. [6, 14]), where the management of uncertainty is based on so-called annotation terms, inspired by Generalized Annotated Logic Programming framework of Kifer and Subrahmanian [10], which is a quite general approach for managing uncertain information.

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.

2 Preliminaries

Description Logics. The specific DL we extend with uncertainty capabilities is $\mathcal{ALC}$, a significant representative of DLs. $\mathcal{ALC}$ is sufficiently expressive to illustrate the main concepts introduced in this paper. So, consider three alphabets of symbols, concept names ($A$), role names ($R$) and individuals, ($a$ and $b$) $^2$. A concept (denoted $C$ or $D$) of the language $\mathcal{ALC}$ is build out from concept names $A$, the top concept $\top$, the bottom concept $\bot$ and according the following syntax rule: if $C$ and $D$ are concepts, then so are $C \cap D$ (concept conjunction), $C \cup D$ (concept disjunction), $\neg C$ (concept negation), $\forall R.C$ (universal quantification) and $\exists R.C$ (existential quantification). A

$^2$Metavariables may have a subscript or a superscript.
terminology, $T$, is a finite set of concept definitions and concept inclusions, called terminological axioms, $\tau$. Let $A$ be a concept name and let $C$ be a concept. A concept definition is an expression of the form $A := C$, while a concept inclusion is an expression of the form $A \sqsubseteq C$. We assume that $T$ is such that no concept name $A$ appears more than once on the left hand side of a terminological axiom $\tau \in T$ and that no cyclic definitions are present in $T$.

An assertion, $\alpha$, 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 Knowledge Base (KB), $\mathcal{K} = (T, \mathcal{A})$, is such that $T$ and $\mathcal{A}$ are finite sets of terminological axioms and assertions, respectively.

An interpretation 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 different individuals into different elements of $\Delta^I$ (called unique name assumption), concept names into subsets of $\Delta^I$ and role names into subsets of $\Delta^I \times \Delta^I$. Note that form a first-order point of view, concept names and role names may be seen as unary predicates and binary predicates, respectively. The interpretation of complex concepts is defined as usual: $\tau^I = \Delta^I$, $\bot^I = \emptyset$, $(C \cap D)^I = C^I \cap D^I$, $(C \cup D)^I = C^I \cup D^I$, $(\neg C)^I = \Delta^I \setminus C^I$, $(\forall R.C)^I = \{d \in \Delta^I : \forall d'(d,d') \notin R^I \text{ or } d' \in C^I\}$ and $(\exists R.C)^I = \{d \in \Delta^I : \exists d'(d,d') \in R^I \text{ and } d' \in C^I\}$. Two concepts $C$ and $D$ are equivalent (denoted $C \equiv D$) when $C^I = D^I$, for all interpretations $I$ (e.g. $\exists R.C \equiv \neg \forall R.(\neg C)$). An interpretation $I$ satisfies $a: C$ (resp. $(a,b): R$) iff $a^I \in C^I$ (resp. $(a^I, b^I) \in R^I$), satisfies $A \sqsubseteq C$ iff $A^I \subseteq C^I$, while satisfies $A = C$ iff $A^I = C^I$. The notions of satisfiability of a terminology, of a set of assertions, of a KB, and that of entailment (denoted $\mathcal{K} \models \alpha$) are as usual. Note that it is harmless to replace a concept inclusion $A \sqsubseteq C$ with a concept definition $A := C \cap A^*$, where $A^*$ is a new primitive concept. Then, as $T$ contains no cycles, we can unfold the concept definitions in $T$, by substituting every concept name occurring in $T$ with its defining term in $T$, obtaining a set of concept definitions, where all the concepts appearing in the right hand sides do not appear in the left hand sides.

Example 1 Consider $\mathcal{K} = (T, \mathcal{A})$, where $T = \{A := \forall R.\neg B\}$, $\mathcal{A} = \{a: \forall R.C\}$, and $\alpha = a:A \cup \exists R.(B \cap C)$. $\mathcal{K} \models \alpha$ holds. In fact, consider a model $I$ of $\mathcal{K}$. Then either $a^I \in A^I$ or $a^I \notin A^I$. In the former case, $I$ satisfies $\alpha$. In the latter case, as $I$ satisfies $T$, $a^I \notin (\forall R.\neg B)^I$, i.e. $a^I \in (\exists R.B)^I$ holds. But, $I$ satisfies $\mathcal{A}$ as well, i.e. $a^I \in (\forall R.C)^I$ and, thus, $a^I \in (\exists R.(B \cap C))^I$. Therefore, $I$ satisfies $\alpha$, which concludes.

Disjunctive Logic Programs. (See, e.g. [6, 14]). Consider an arbitrary first order language that contains infinitely many variable symbols, finitely many constants, and predicate symbols, but no function symbols. A term is either a constant or a variable. An atom is of the form $p(l_1, \ldots, l_n)$, where all $l_i$ are terms and $p$ is an $n$-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 an expression of the form $\text{not}(l)$, where $l$ is a literal. A ground extended literal is an extended literal without variables. For a set $X$ of extended literals, $X^- = \{\text{not}(l) \mid \text{not}(l) \in X\}$, while $\neg X = \{-l \mid l \in X\}$, where we define $\neg \neg a = a$.

---

3We say that $A$ directly uses primitive concept $B$ in $T$, if there is $\tau \in T$ such that $A$ is on the left hand side of $\tau$ and $B$ occurs in the right hand side of $\tau$. Let uses be the transitive closure of the relation directly uses in $T$. $T$ is cyclic iff there is $A$ such that $A$ uses $A$ in $T$.  

3
A disjunctive logic program \( P \), is a finite set of rules of the form \( \gamma \leftarrow \delta \), where \( \gamma \) and \( \delta \) are finite sets of extended literals. For easy, we may omit the graph brackets in a rule. We call programs where for each rule \( \gamma \cup \delta = \emptyset \) programs without negation as failure (\( \text{naf} \)). Programs, where in each rule \( |\gamma| = 1 \) are called normal. Programs without \( \text{naf} \), containing positive literals only, are called positive. Programs that do not contain variables are called ground. For a program \( P \), and a (possibly infinite) non-empty set of constants \( H \), such that every constant appearing in \( P \) is in \( H \), we call \( P_H \) the grounded program obtained from \( P \) by substituting every variable in \( P \) by every possible constant in \( H \). Note that \( P_H \) may contain an infinite number of rules (if \( H \) is infinite). The universe of a grounded program \( P \) is the (possibly infinite) non-empty set of constants \( H_P \) appearing in \( P \). Note that \( H_{P_H} = H \). The base of a grounded program \( P \) is the (possibly infinite) set \( B_P \) of ground atoms that can be constructed using the predicate symbols in \( P \) with the constants in \( H_P \).

An interpretation \( I \) of a grounded program \( P \) is any consistent set of literals being a subset of \( B_P \cup \lnot B_P \). Furthermore, we say that \( I \) satisfies an extended literal \( \text{not}(l) \) iff \( I \) does not satisfy \( l \). An interpretation \( I \) of a grounded program \( P \) without \( \text{naf} \) satisfies a rule \( \gamma \leftarrow \delta \) iff \( \gamma \cap \delta \neq \emptyset \) whenever \( \delta \subseteq I \). An interpretation \( I \) is a model of program \( P \) without \( \text{naf} \) iff it satisfies every rule in \( P \). \( I \) is a minimal model of \( P \) iff \( I \) is a model of \( P \) and there is no model \( J \subset I \) of \( P \). For a grounded program \( P \) and an interpretation \( I \), the Gelfond-Lifschitz transformation [6, 14], is the program \( P^I \) without \( \text{naf} \), obtained by deleting in \( P \) (i) each rule that has \( \text{not}(l) \) in its body with \( l \in I \); (ii) each rule that has \( \text{not}(l) \) in its head with \( l \not\in I \); and (iii) all \( \text{not}(l) \) in the bodies and heads of the remaining rules.

Finally, an interpretation of a program \( P \) (possibly not grounded) is a pair \( I = (H, I) \), such that \( I \) is an interpretation of the grounded program \( P_H \). An interpretation \( I = (H, I) \) of a program \( P \) is a stable model of \( P \) iff \( I \) is a minimal model of \( P_H \). It can easily be shown that, if \( I = (H, I) \) is a stable model of \( P \), then \( I \) is a model of \( P_H \) \(^4\). We say that a program \( P \) entails a ground extended literal \( l \), denoted \( P \models l \) iff every stable model of \( P \) satisfies \( l \). Note that \( P \models \text{not}(l) \), where \( l \) is a literal if every stable model satisfying \( P \) does not satisfy \( l \).

Example 2 Consider \( P = \{(p, \neg q \leftarrow \text{not}(-r)), (r \leftarrow \neg s), (t \leftarrow p), (t \leftarrow \neg q)\} \). \( P \) has two stable models \( I_1 = \{p, t\} \) and \( I_2 = \{\neg q, t\} \), so \( P \models t \), while \( P \models \text{not}(r) \). Note that the program \( \{a \leftarrow \text{not}(a)\} \) has no stable model.

Negation \( \neg \) behaves like explicit negation in the sense that negative literals should explicitly be derived. It is well known that we can replace negative literals with their ‘positive form’. Consider for each \( n \)-ary predicate symbol \( p \), a new \( n \)-ary predicate symbol \( \bar{p} \). For a literal \( l \), its positive form, \( \bar{l} \), is \( \bar{l} = l \), if \( l \) is a positive literal, is \( \bar{l} = \bar{p}(t_1, \dots, t_n) \) if \( l \) is a negative literal \( \neg p(t_1, \dots, t_n) \). With \( P \), we indicate the positive form of \( P \) obtained from \( P \) by replacing every literal occurring in \( P \) with its positive form. The definition of \( X \), where \( X \) is a set of extended literals is similar. Then, an interpretation \( I \) is a stable model of a program \( P \) iff \( I \) is a stable model of \( P \) \(^6\).


\(^4\)Note that \( P_H \) contains grounded extended literals

\(^5\)The term ‘description logic program’ has already been used in [7], but with a slightly different meaning.

\(^6\)A Description Logic Program (DLP) is a pair \( \mathcal{D} P = (\mathcal{T}, \mathcal{P}) \), where \( \mathcal{T} \) is a terminology and \( \mathcal{P} \) is a disjunctive logic program \(^5\). Roles names and concept names appearing in \( \mathcal{T} \) may appear in the body \( \beta \) of a rule.
γ ← δ ∈ P and are managed as unary and binary predicates, respectively. We allow the following exception for representing assertions. An assertion \( a : A \), where \( A \) is a concept name is represented by means of the rule \( A(a) ← \), while an assertion \( (a, b) : R \) is represented by means of the rule \( R(a, b) ← \). Note that we do not allow concept and role names that appear in \( T \) to appear in the head of the rules (except for representing assertions) 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.

An interpretation is a model of \( DP = (T, P) \) is a pair \( I = (H, I) \), where \( I \) is an interpretation for \( P \) and \( (\Delta^2, \cdot) \) is an interpretation for \( T \), where \( \Delta^2 = H \), and for concept names \( A \) and roles names \( R, A^2 = \{a | A(a) \in I \} \) and \( R^2 = \{(a, b) | R(a, b) \in I \} \), respectively. An interpretation is a model of \( DP = (T, P) \) if it is a model of \( T \) and a stable model of \( P \). The definition of entailment is as usual.

Example 3. Consider \( DP = (T, P) \), with \( T = \{A_1 : \forall R, \neg C, B_1 : \forall R, D, E : = C \cap D\} \) and \( P = \{(p(X) ← A(X)), (p(X) ← R(X, Y), E(Y), \text{not}(r(y))), (B(a) ← \})\}. Then, by reasoning like in Example 1, it follows that \( DP \models p(a) \). In fact, \( DP \) has two models with universe \( H = \{a\} \) and \( I_1 = \{B(a), A(a), p(a)\} \) and \( I_2 = \{B(a), R(a, a), C(a), D(a), E(a), p(a)\} \).

Decision procedures for DLPs. In the usual combination of logic programs, more precisely Horn rules, with description logics, like e.g. [5, 13], the reasoning algorithms are a combination of both a description logic reasoner and a logic program reasoner. For instance, according to [13], one usually computes the so-called canonical models of the terminological component together with the facts involving concepts and roles, then translates these facts into horn facts and asks whether for each canonical models of the terminological component together with the facts involving concepts and roles, then there is a refutation for the goal. This approach hardly applies to our case as negation-as-failure is present and rules heads are disjunctions.

We follow rather a different approach. We use a recent result [8], where it is shown that for a terminology w.r.t. the highly expressive description logic \( SHIQ^\ast \), containing qualified number restrictions, inverse roles and transitive role closure, a model preserving translation into a disjunctive logic program can be given. This has the considerable practical advantage that deciding an entailment problem in DLPs may be deferred to a disjunctive logic program reasoner like DLV [4] or smodels [16]. The method is as follows. Consider \( DP = (T, P) \). The closure, \( \text{closure}(DP) \), of \( DP \) is defined as follows. For every concept expression \( D \) and primitive role \( R \) in \( DP \) we have \( \{D, R\} \subseteq \text{closure}(DP) \) and for every \( D \) in \( \text{closure}(DP) \), we have one of the following:

\( (i) \) if \( D = \neg D' \), then \( D' \in \text{closure}(DP) \); \( (ii) \) if \( D = D' \cap D'' \), then \( \{D', D''\} \subseteq \text{closure}(DP) \); \( (iii) \) if \( D = D' \cup D'' \), then \( \{D', D''\} \subseteq \text{closure}(DP) \); \( (iv) \) if \( D = \exists R.D' \), then \( \{R, D'\} \subseteq \text{closure}(DP) \); \( (v) \) if \( D = \forall R.D' \), then \( \{D', \exists R.\neg D'\} \subseteq \text{closure}(DP) \); and \( (vi) \) for all \( D \in \text{closure}(DP) \), \( \neg D \in \text{closure}(DP) \). Let \( \Omega(DP) \) be the disjunctive
logic program, obtained from $\mathcal{DP}$, as follows.

<table>
<thead>
<tr>
<th>closure$(\mathcal{DP})$</th>
<th>$\Omega(\mathcal{DP})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>concept name $A$</td>
<td>$A(X), \text{not}(A(X)) \leftarrow$</td>
</tr>
<tr>
<td>role name $R$</td>
<td>$R(X,Y), \text{not}(R(X,Y)) \leftarrow$ $\sim R(X,Y), \text{not}(\sim R(X,Y)) \leftarrow$</td>
</tr>
<tr>
<td>expressions $D$</td>
<td>$D' = \neg D'$ $D = D' \cap D''$ $D = D' \cup D''$ $D = \exists R.D'$ $D = \forall R.D'$ $A = C \in T$</td>
</tr>
<tr>
<td></td>
<td>$\sim D'(X) \leftarrow \text{not}(D'(X))$ $(D' \cap D'')(X) \leftarrow D'(X), D''(X)$ $(D' \cup D'')(x) \leftarrow D'(x), D''(x)$ $(\exists R.D')(X) \leftarrow R(X,Y), D'(Y)$ $(\forall R.D')(X) \leftarrow (\exists R, \sim D')(X)$ $A(X), \text{not}(C(X))$ $\leftarrow \text{not}(A(X)), C(X)$</td>
</tr>
</tbody>
</table>

Let $\mathcal{P}_{\mathcal{DP}} = \mathcal{P} \cup \Omega(\mathcal{DP})$, then the following theorem follows easily from [8].

**Theorem 1** An interpretation $I$ of $\mathcal{DP} = (T, \mathcal{P})$ is a model of $\mathcal{DP}$ iff $I$ is a model of $\mathcal{DP}' = (\emptyset, \mathcal{P}_{\mathcal{DP}})$, i.e. $I$ is a model of the disjunctive logic program $\mathcal{P}_{\mathcal{DP}}$.

# 3 Description Logic Programs with Uncertainty

We are going now to define an extension of DLPs towards the management of uncertainty. In particular, we will integrate the so-called Generalized Annotated Logic Programs (GAP) framework of Kifer and Subrahmanian [10] into the DLP framework. GAPs are an unifying framework for many existing approaches towards the management of uncertainty in logic programs. Informally, in GAP a rule is of the form $a_0 \cdot a_0 \leftarrow a_1 \cdot \mu_1, \ldots, a_n \cdot \mu_n$, where the $a_i$ are atoms and the $\mu_i$ are either certainty values taken from a lattice $\mathcal{C}$, e.g. $[0, 1]$, or variables ranging over certainty values, or a computable function $f: \mathcal{C}^m \rightarrow \mathcal{C}$. An example of rule is $p: x + y - xy \leftarrow q: x, r: x$, whose intended semantics is to specify how to compute the certainty value of the atom $p$ in the head, from the certainty values of the atoms $q$ and $r$ in the body. For instance, for the GAP with rules $(q: 0.3 \leftarrow)$, $(r: 0.4 \leftarrow)$ and $(p: x + y - xy \leftarrow q: x, r: x)$ we conclude $p: 0.58$. Of course, in order to cope with this extension to logic programs, we have to extend the description logic part adequately, as well. In the following we first present the description logic component with uncertainty management and then the logic program component with uncertainty.

Furthermore, for ease the presentation, we fix the certainty lattice to be the unit interval $[0, 1]$.

**Uncertainty in ALC.** Our uncertainty description logic is the one presented in [17], which we call here $\mu\text{ALC}$. The main idea is that an assertion $a: C$, rather being interpreted as either true or false, will be mapped into a certainty value $c \in [0, 1]$. The intended meaning is that $c$ indicates to which extend (how certain it is that) ‘$a$ is a $C$’. Similarly for role names. Formally (see [17] for the details), a $\mu$interpretation is now a pair $\mathcal{I} = (\Delta^\mathcal{I}, \mathcal{T}^\mathcal{I})$, where $\Delta^\mathcal{I}$ is the domain, whereas $\mathcal{T}^\mathcal{I}$ is an interpretation function mapping (i) individuals as for the classical case; (ii) a concept $C$ into a function $C^\mathcal{I}: \Delta^\mathcal{I} \rightarrow [0, 1]$; and (iii) a role $R$ into a function $R^\mathcal{I}: \Delta^\mathcal{I} \times \Delta^\mathcal{I} \rightarrow [0, 1]$. The interpretation function $\mathcal{T}^\mathcal{I}$ has to satisfy the following equations: for all $d \in \Delta^\mathcal{I}$, $\mathcal{T}^\mathcal{I}(d) = 1$, 

\( \bot^2(d) = 0, \) \( (C \cap D)^2(d) = \min(C^2(d), D^2(d)), \) \( (C \cup D)^2(d) = \max(C^2(d), D^2(d)), \) 
\( (-C)^2(d) = 1 - C^2(d), \) \( (\forall R.C)^2(d) = \inf_{d' \in \Delta^X} \{\max(1 - R^2(d, d'), C^2(d'))\}, \) and \( (\exists R.C)^2(d) = \sup_{d' \in \Delta^X} \{\min(R^2(d, d'), C^2(d'))\}. \) The definition of concept equivalence is like for ALC. Two concepts \( C \) and \( D \) are equivalent iff \( C^2 = D^2, \) for all \( \mu \) interpretations \( I. \) As for classical ALC, dual relationships between concepts hold: e.g. \( (C \cap D) \equiv \neg(-C \cup -D) \) and \( (\forall R.C) \equiv \neg(\exists R.\neg C), \) but \( A \not\equiv B \cap (-B \cup A). \)

A \textit{massertion} (denoted \( \mu a \)) is an expression \( (\alpha \geq c) \) or \( (\alpha \leq c'), \) where \( \alpha \) is an ALC assertion and \( c, c' \in [0, 1]. \) From a semantics point of view, a massertion \( (\alpha \geq c) \) constrains the certainty value of \( \alpha \) to be less or equal to \( c \) (similarly for \( \geq \)). An \( \mu \) interpretation \( I \) \textit{satisfies} \( (a; C \geq c) \) (respectively \( (a; b; R \geq c)) \) iff \( C^2(a^2) \geq c \) (respectively \( R^2(a^2, b^2) \geq c \)). Similarly for \( \leq \). Note that, e.g. \( (a; \neg C \geq c) \) and \( (a; C \leq 1 - c) \) are satisfied by the same set of \( \mu \) interpretations. Concerning terminological axioms, an \( \mu \) interpretation \( I \) \textit{satisfies} \( A \subseteq C \) iff \( \forall d \in \Delta^2, A^2(d) \leq C^2(d), \) while \( I \) \textit{satisfies} \( A = C \) iff \( \forall d \in \Delta^2, A^2(d) = C^2(d). \) A \textit{Knowledge Base} (\( \mu KB \)) is pair \( \mu K = (T, \mu A), \) where \( T \) and \( \mu A \) are finite sets of terminological axioms and massertions, respectively. The notions of \textit{satisfiability (model)} of a \( \mu KB \) and that of \textit{entailment} are as usual. Finally, given \( \mu K \) and an assertion \( \alpha, \) it is of interest to compute \( \alpha \)'s best lower and upper certainty value bounds. The \textit{greatest lower bound} of \( \alpha \) w.r.t. \( \mu K \) (denoted glb(\( \mu K, \alpha \))) is \( \sup\{c : \mu K \models (\alpha \geq c)\}, \) while the \textit{least upper bound} of \( \alpha \) with respect to \( \mu K \) (denoted lub(\( \mu K, \alpha \))) is \( \inf\{c : \mu K \models (\alpha \leq c)\} \) (sup \( \emptyset = 0, \inf \emptyset = 1). \)

Determining the lub and the glb is called the \textit{Best Certainty Value Bound (BCVB)} problem. In [17] decision procedures for the satisfiability, the entailment and the BCVB problem are given. So we do not further investigated them here.

**Example 4** Similarly to Example 1, consider \( \mu K = (T, \mu A), \) with \( T = \{A = \forall R.\neg B\}, \) \( \mu A = \{(a; \forall R.C \geq 0.7)\}. \) For \( \alpha = a; A \cup \exists R. (B \cap C), \) glb(\( \mu K, \alpha \)) = 0.5 and lub(\( \mu K, \alpha \)) = 1 hold.

**Uncertainty in disjunctive logic programs.** Our extension of disjunctive logic programs with uncertainty is based on [10, 15]. Alternative, quite general approaches, like [11], can be worked out similarly.

An \textit{annotation function} of arity \( n \) is a total and computable function \(^6 f : ([0, 1])^n \rightarrow [0, 1]. \) Assume a new alphabet of \textit{annotation variables}, which will denote a value in \([0, 1]\) and can only appear in so-called \textit{annotation terms}. An \textit{annotation item}, \( \nu, \) is one of the following: \( (i) \) a real \( c \in [0, 1], \) or an annotation variable, or \( (ii) \) or of the form \( f(\nu_1, \ldots, \nu_n), \) where \( f \) is an \( n \)-ary annotation function and all \( \nu_i \) are annotation items. An \textit{annotation term}, \( \lambda, \) is of the form \( [\nu, \nu'] \), where \( \nu \) and \( \nu' \) are annotation items. Annotation terms are supposed to denote subintervals of \([0, 1]. \)

Let \( l \) be a literal and \( \lambda \) an annotation term. A \textit{pliteral}, denoted \( \mu l, \) is of the form \( l; \lambda. \) The intended meaning is that “the certainty of \( l \) lies in the interval \( \lambda. \) An \textit{extended pliteral} is of the form not(\( \mu l \)), where \( \mu l \) is a pliteral. The intended meaning of not(\( l; \lambda \)) is that “\( l \) is not provable that the certainty of \( l \) lies in the interval \( \lambda. \) A \textit{μdisjunctive logic program}, \( \mu P, \) is a finite set of \( μ rules \) of the form \( γ → δ, \) where \( γ \) and \( δ \) are finite sets of extended pliters. In order to avoid straightforward repetition, if not stated otherwise, definitions related to \( \mu \)disjunctive logic programs, parallels those for disjunctive logic programs. Furthermore, in grounding a \( μ \)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 \( f(\nu_1, \ldots, \nu_n) \) are replaced with

\(^6\)The result of \( f \) is computable in a finite amount of time.
the result of the computation of \( f(\nu_1, \ldots, \nu_n) \). Note that a grounded program \( \mu \mathcal{P} \) may contain an infinite number of rules due to the grounding of annotation terms. For a grounded program \( \mu \mathcal{P} \), \( B_{\mu \mathcal{P}} \) is the set of ground atoms \( a \) that can be constructed using the predicate symbols in \( \mu \mathcal{P} \), where \( a \) is grounded with the constants in \( H_{\mu \mathcal{P}} \) (annotations terms are not considered).

An interpretation \( I \) of a grounded program \( \mu \mathcal{P} \) is any (possibly partial) function \( I: B_{\mu \mathcal{P}} \to [0,1] \) (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 \( \neg a: [\nu_1, \nu_2] \) with \( a: [\neg \nu_2, \neg \nu_1] \). Like for disjunctive logic programs, we say that \( I \) satisfies every rule in some \( \neg \mathit{af} \) does not satisfy \( l \): \( I(l) \notin \lambda \). An \( \mu \)-interpretation \( I \) of a grounded program \( \mu \mathcal{P} \) without \( \neg \mathit{af} \) satisfies a \( \mu \) literal \( \lambda \) iff it satisfies every \( \mu \) literal \( \lambda I \) in \( \gamma \). A \( \mu \)-interpretation \( I \) is a \( \mu \) model of program \( \mu \mathcal{P} \) without \( \neg \mathit{af} \) iff it satisfies every rule in \( \mu \mathcal{P} \).

**Example 5** Consider the grounded program \( \mu \mathcal{P} \) without \( \neg \mathit{af} \), \( \{ (c, [0.1, 0.3]) \rightarrow b, (a, [0.2, 0.7]), (b, [0.3, 0.6]) \} \). Let us consider the following two partial functions \( \mu \mathcal{F}_1 \) and \( \mu \mathcal{F}_2 \), assigning to atoms intervals and defined as follows: \( \mu \mathcal{F}_1(a) = [0.2, 0.7], \mu \mathcal{F}_1(c) = [0.1, 0.3], \mu \mathcal{F}_2(b) = [0.3, 0.6]. \mu \mathcal{F}_2(c) = [0.1, 0.3] \). It is easily verified that for each \( \mu \) model \( I \) of \( \mu \mathcal{P} \), we have that for some \( i = 1,2 \), \( \text{def}(\mu \mathcal{F}_i) \subseteq \text{def}(I) \) and for all ground atoms \( p \in \text{def}(\mu \mathcal{F}_i), I(p) \in \mu \mathcal{F}_i(p) \). Essentially, \( \mu \mathcal{F}_1 \) and \( \mu \mathcal{F}_2 \) may be seen as a partitioning of the \( \mu \) models, such that \( \mu \mathcal{F} \) is 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 interpretation partitioning. 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 \( \sigma_1 \preceq \sigma_2 \) iff \( \sigma_2 \subseteq \sigma_1 \). Similarly, \( \sigma_1 \preceq \sigma_2 \) iff \( \sigma_2 \subseteq \sigma_1 \). 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 \( \mathcal{F} \) of a grounded program \( \mu \mathcal{P} \) without \( \neg \mathit{af} \) is a (possibly partial) function \( \mathcal{F}: B_{\mu \mathcal{P}} \to C[0,1] \). An interval interpretation \( \mathcal{F} \) is a representative of a whole family of \( \mu \) interpretations \( I \): we write \( I \in \mathcal{F} \) iff \( \text{def}(I) = \text{def}(\mathcal{F}) \) and for all \( a \in \text{def}(\mathcal{F}), I(a) \in \mathcal{F}(a) \) (see Example 5). We extend interval interpretations \( \mathcal{F} \) to ground literals \( l = \neg a \) as usual: \( \mathcal{F}(l) = \neg \mathcal{F}(a) \). For interval interpretations \( \mathcal{F}_1 \) and \( \mathcal{F}_2 \), \( \mathcal{F}_1 \preceq \mathcal{F}_2 \) iff \( \text{def}(\mathcal{F}_2) \subseteq \text{def}(\mathcal{F}_1) \) and for all \( a \in \text{def}(\mathcal{F}_2), \mathcal{F}_1(a) \preceq \mathcal{F}_2(a) \). Similarly, \( \mathcal{F}_1 \preceq \mathcal{F}_2 \) iff \( \text{def}(\mathcal{F}_1) \subseteq \text{def}(\mathcal{F}_2) \) and for some \( a \in \text{def}(\mathcal{F}_1), \mathcal{F}_1(a) \preceq \mathcal{F}_2(a) \). A \( \preceq \) least interval interpretation, \( \mathcal{F}_1 \), assigns to all ground atoms in \( B_{\mu \mathcal{P}} \) the interval \( [0,1] \), meaning essentially that the certainty of the atoms is unknown. An interval interpretation \( \mathcal{F} \) satisfies a ground \( \mu \) literal \( l: \lambda \) iff \( \lambda \preceq \mathcal{F}(l) \). An interval interpretation \( \mathcal{F} \) of a grounded program \( \mu \mathcal{P} \) without \( \neg \mathit{af} \) satisfies a rule \( \gamma \rightarrow \delta \) iff \( \mathcal{F} \) satisfies every \( \mu \) literal in \( \delta \) then \( \mathcal{F} \) satisfies some \( \mu \) literal in \( \gamma \). An interval interpretation \( \mathcal{F} \) is an interval model of program \( \mu \mathcal{P} \) without \( \neg \mathit{af} \) iff it satisfies every rule in \( \mu \mathcal{P} \).

Furthermore, \( \mathcal{F} \) is minimal as well iff there is no interval model \( \mathcal{F}' \prec \mathcal{F} \) of \( \mu \mathcal{P} \). For instance, in Example 5, \( \mathcal{F}_1 \) and \( \mathcal{F}_2 \) are the only two minimal interval models of \( \mu \mathcal{P} \). Like in [10], it can be shown that if \( \mu \mathcal{P} \) is normal grounded program without \( \neg \mathit{af} \), then there is an unique minimal interval model for \( \mu \mathcal{P} \), which is the \( \preceq \) least fixed-point, \( \mathcal{F}(\mu \mathcal{P}) \), of the following \( T_{\mu \mathcal{P}} \) monotone operator: for all \( a \in B_{\mu \mathcal{P}}, \)
Given a grounded µ-program (possibly with naf) M P and an µinterpretation I for M P, the Gelfond-Lifschitz transformation, is the grounded positive µ-program M P′, obtained by deleting in M P, (i) each rule that has not(µ) in its body and I satisfies µl; (ii) each rule that has not(µ) in its head and I does not satisfy µl; and (iii) all not(µ) in the bodies and heads of the remaining rules. Finally, an µinterpretation of a µrogram P (possibly not grounded) is a pair I = (H, I), such that I is an µinterpretation of the grounded program M P | H. An µinterpretation I = (H, I) of a µrogram M P is a stable µodel of µ P iff I ∈  I for a minimal interval model  of M P | H. Finally, we say that a program M P entails a grounded extended µetal, denoted M P |= µl iff every stable µodel of M P satisfies µl. For instance, in Example 5, we have that any stable µodel I of µ P is such that I ∈  I or I ∈  2.

Example 6 Consider the following µ-programs M P1, M P2, M P3 and M P4, where M P1 = {r1, r2}, M P2 = {r1, r3}, M P3 = {r1, r4}, M P4 = {r1, r5} and the rules r1, r2, r3, r4, r5 are (r1 : a : [0, 0.8] ← ), (r2 : b : [0.4, 0.5] ← not(a: [0, 0.2, 0.3])), (r3 : b : [0.4, 0.5] ← not(a: [0.2, 0.7])), (r4 : b : [x, y] ← a: [x, y]) and (r5 : b : [x, y] ← not(a: [x, y])). It can be verified that for any stable µodel I of M P, we have that (i) for M P1, I(a) ∈ [0.6, 0.8] and I(b) ∈ [0.4, 0.5]. Therefore, M P1 |= b: [0.4, 0.5]; (ii) for M P2, I(a) ∈ [0.6, 0.8] and if I(a) ∈ [0.7, 0.8] then I(b) ∈ [0.4, 0.5] else I is undefined on b. Therefore, M P2 ̸|= b: [c, c′], for any c, c′ ∈ [0, 1]. But, M P2 |= not(b: [c, c′]) if [c, c′] ∩ [0, 0.4, 0.5] = Φ; and (iii) for M P3, I(a) ∈ [0.6, 0.8] and I(b) ∈ [0.6, 0.8]. Therefore, M P3 |= b: [0.6, 0.8]. For M P4, first note that for any I satisfying r1, I(a) ∈ [0.6, 0.8] holds. Therefore, M P4 = {r1} ∪ {b: [c, c′] ← } I(a) ̸∈ [c, c′], c ≤ c′ ∈ [0, 1]}, whose least interval model  is such that  I(a) = [0.4, 0.5] and  I(b) = Φ. So, M P4 has no stable model.

Uncertainty in Description Logic Programs. A µDescription Logic Program (µDLP), denoted M DLP, is a pair (T, M P), where T is a µALC terminology and M P is a µrogram, where roles and concept names appearing in T may appear also in the body of the rules, and metassertions of the form ⟨a:A ∧ c⟩, where A is a concept name, are represented by means of the rule A(a): [c, 1] ←. Similarly for ⟨a:A ≤ c⟩, ⟨a,b⟩: R ≥ c) and ⟨a,b⟩: R ≤ c.

An interpretation for a µDLP M DLP is a pair I = (H, I), where I is a µinterpretation for the µrogram M P and (Δ I) is a µinterpretation for the terminology T, where Δ I = H, and for concept names A and roles names R, A′(a) = I(A(a)) and R′(a) = I(R(a, b)), respectively. An interpretation for a µDLP M DLP is a µodel of a µDLP M DLP = (T, M P) iff I is a µodel of T and a stable µodel of M P. Entailment is defined as usual.

Example 7 Let us consider the DLP of Example 3, but extended with uncertainty as follows. M DLP = (T, P), where T = {A: ∀R.¬C, B: ∀R.D, E: C ⊓ D} and
Consider the atom $p(a)$. Then, by reasoning by cases like in Example 3, it follows that $\mu DP \models p(a)\{0.5, 1\}$. In fact, the $\mu$ models $I = (H, I)$ of $\mu DP$ are such that $H = \{a\}$ and either $\text{def}(I) = \{B(a), A(a), p(a)\}$ or $\text{def}(I) = \{B(a), R(a), A(a), D(a), E(a), p(a)\}$. 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 $\mu DP$, we have $I(p(a)) \geq 0.5$.

**Reasoning in $\mu$DLP.** Reasoning in $\mu$DLP is not an easy task. From computational point of view, several levels of complexity come to play a significant role: the computational complexity of the description logic component, the one from the logic programming component and the one from the literal annotation part. Due to the expressive power of $\mu$DLP it is not surprising that even for a small fragment of it, we have negative results known from the literature. For instance, in [13] it is shown, that for a set of recursive Horn rules and just the $\forall R.C$ constructor, the entailment problem is undecidable, even without naf and without managing uncertainty. Similarly, the simple example showing the non-continuity of the $T_{\mu p}$ is indicating a similar result for normal $\mu$ programs without naf. In the general case, an extension of the existential entail algorithm [13] to our uncertainty setting setting seems not to be an easy task, as there is yet no resolution method (top-down method) known for disjunctive logic programs. On the other hand, an approach following [8], where e.g. we use a resolution method (top-down method) known for disjunctive logic programs. On the other hand, an approach following [8], where e.g. we use a resolution method (top-down method) known for disjunctive logic programs.
not((pσ)\((X_1, \ldots, X_n)\)); and \((v) \leftarrow (pσ_1)\((X_1, \ldots, X_n)\), \((p\bar{σ}_2)\((X_1, \ldots, X_n)\) if \(σ_1 \subseteq σ_2\). \(\bar{p}\) is a new symbol associated to \(p\). \(\bar{p}σ(\cdot)\) encodes the fact that the certainty of \(p(\cdot)\) is not in \(σ\). To the resulting disjunctive logic program, we can apply a standard disjunctive logic program reasoner to compute all stable models and check then for entailment (e.g., for \(\mu\mathcal{DP}_2\) in Example 6, we get two stable models \(I_1 = \{a[0.6, 0.8], a[0.2, 0.7], a[0.6, 0.7], \ldots\}\) and \(I_2 = \{a[0.6, 0.8], \tilde{a}[0.2, 0.7], b[0.4, 0.5], \ldots\}\), meaning that for any stable model \(I\) of \(\mu\mathcal{DP}_2\) if \(I(a) \in [0.7, 0.8]\) then \(I(b) \in [0.4, 0.5]\) else \(I\) is undefined on \(b\).)

Note that, from Theorem 2 it follows that \(\mu\mathcal{ALC}\) [17] can be translated into disjunctive logic programs as well. Furthermore, Theorem 2 can be generalized to the case where the iterated application of annotation terms to \(N\) \(\mathcal{DP}\) generates a finite set, or where the \(\mu\) \(\mathcal{DLP}\) does not contain recursive rules.

4 Conclusions

We integrated the management of uncertainty into a highly expressive family of representation languages, called \(\mu\) \(\mathcal{DLP}\), resulting from the combination of description logics and disjunctive logic programs. We defined syntax, semantics and discussed some reasoning issues related to \(\mu\) \(\mathcal{DLP}\). Our motivation is inspired by its application to the integration of heterogeneous information sources and distributed search in the Semantic Web, where uncertain mappings between concepts of different ontologies may be required.

The main topic of our future work consists in addressing the computational issue more extensively, especially investigating how already existing reasoning techniques may applied to either full \(\mu\) \(\mathcal{DLPs}\) or to meaningful fragments of them.

References


