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* (μDLPs). 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]

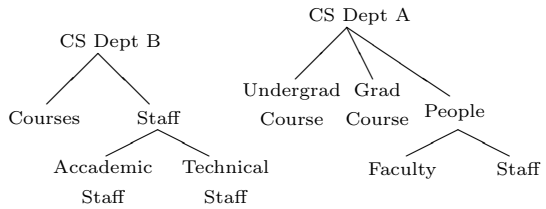
Terms: Theory

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].

[1] www.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 $C \sqcap D$ (concept conjunction), $C \sqcup D$ (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, $\mathcal{T}$, is a finite set of concept definitions and concept inclusions, called *terminological axioms*, τ. 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 $\mathcal{T}$ is such that no concept name A appears more than once on the left hand side of a terminological axiom $\tau \in \mathcal{T}$ and that no cyclic definitions are present in $\mathcal{T}$ [3]. An *assertion*, α, is an expression $a{:}C$ ("a is an instance of C"), or an expression $(a,b){:}R$ ("(a,b) is an instance of R"). A *Knowledge Base* (KB), $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, is such that $\mathcal{T}$ and $\mathcal{A}$ are finite sets of terminological axioms and assertions, respectively.

An *interpretation* is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consisting of a non empty set $\Delta^{\mathcal{I}}$ (called the *domain*) and of an *interpretation function* $\cdot^{\mathcal{I}}$ mapping different individuals into different elements of $\Delta^{\mathcal{I}}$ (called *unique name assumption*), concept names into subsets of $\Delta^{\mathcal{I}}$ and role names into subsets of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. 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: $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$, $\bot^{\mathcal{I}} = \emptyset$, $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$, $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$, $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$, $(\forall R.C)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} : \forall d'.(d,d') \notin R^{\mathcal{I}} \text{ or } d' \in C^{\mathcal{I}}\}$ and $(\exists R.C)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} : \exists d'.(d,d') \in R^{\mathcal{I}} \text{ and } d' \in C^{\mathcal{I}}\}$. Two concepts C and D are *equivalent* (denoted $C \equiv D$) when $C^{\mathcal{I}} = D^{\mathcal{I}}$, for all interpretations $\mathcal{I}$ (e.g. $\exists R.C \equiv \neg\forall R.\neg C$). An interpretation $\mathcal{I}$ *satisfies* $a{:}C$ (resp. $(a,b){:}R$) iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$ (resp. $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$), *satisfies* $A \sqsubseteq C$ iff $A^{\mathcal{I}} \subseteq C^{\mathcal{I}}$, while *satisfies* $A{:=}\,C$ iff $A^{\mathcal{I}} = C^{\mathcal{I}}$. 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 $A \sqsubseteq C \in \mathcal{T}$ with a concept definition $A{:=}\,C \sqcap A^*$, where A^* is a new primitive concept. Then, as $\mathcal{T}$ contains no cycles, we can *unfold* the concept definitions in $\mathcal{T}$, by substituting every concept name occurring in $\mathcal{T}$ with its defining term in $\mathcal{T}$, 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* $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$*, where* $\mathcal{T} = \{A{:=}\,\forall R.\neg B\}$*,* $\mathcal{A} = \{a{:}\forall R.C\}$*, and* $\alpha = a{:}A \sqcup \exists R.(B \sqcap C)$*.* $\mathcal{K} \models \alpha$ *holds. In fact, consider a model* $\mathcal{I}$ *of* $\mathcal{K}$*. Then either* $a^{\mathcal{I}} \in A^{\mathcal{I}}$ *or* $a^{\mathcal{I}} \notin A^{\mathcal{I}}$*. In the former case,* $\mathcal{I}$ *satisfies* α*. In the latter case, as* $\mathcal{I}$ *satisfies* $\mathcal{T}$*,* $a^{\mathcal{I}} \notin (\forall R.\neg B)^{\mathcal{I}}$*, i.e.* $a^{\mathcal{I}} \in (\exists R.B)^{\mathcal{I}}$ *holds. But,* $\mathcal{I}$ *satisfies* $\mathcal{A}$ *as well, i.e.* $a^{\mathcal{I}} \in (\forall R.C)^{\mathcal{I}}$ *and, thus,* $a^{\mathcal{I}} \in (\exists R.(B \sqcap C))^{\mathcal{I}}$*. Therefore,* $\mathcal{I}$ *satisfies* α*, 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 $p(t_1, \ldots, t_n)$, 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 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 $\neg X = \{\neg l \mid l \in X\}$, where we define $\neg\neg a = a$.

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

A *disjunctive logic program* $\mathcal{P}$, is a finite set of *rules* of the form $\gamma \leftarrow \delta$, where γ and δ 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 (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*. For a program $\mathcal{P}$, and a (possibly infinite) non-empty set of constants H, such that every constant 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 constant 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 constants $H_\mathcal{P}$ appearing in $\mathcal{P}$. Note that $H_{\mathcal{P}_H} = H$. The *base* of a grounded program $\mathcal{P}$ is the (possibly infinite) set $\mathcal{B}_\mathcal{P}$ of ground atoms that can be constructed using the predicate symbols in $\mathcal{P}$ with the constants in $H_\mathcal{P}$.

An *interpretation* I of a grounded program $\mathcal{P}$ is any consistent set of literals being a subset of $\mathcal{B}_\mathcal{P} \cup \neg\mathcal{B}_\mathcal{P}$. Furthermore, we say that I *satisfies* an extended literal not(l) iff I does not satisfy l. 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 iff it satisfies every rule in $\mathcal{P}$. I is a *minimal model* of $\mathcal{P}$ iff 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 [6, 14], is the program $\mathcal{P}^I$ without naf, obtained by deleting in $\mathcal{P}$, (i) each rule that has not(l) in its body with $l \in I$; (ii) each rule that has not(l) in its head with $l \notin I$; and (iii) all 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 $\mathcal{I} = (H, I)$, such that I is an interpretation of the grounded program $\mathcal{P}_H$. An interpretation $\mathcal{I} = (H, I)$ of a program $\mathcal{P}$ is a *stable model* of $\mathcal{P}$ iff I is a minimal model of $\mathcal{P}_H^I$. It can easily be shown that, if $\mathcal{I} = (H, I)$ is a stable model of $\mathcal{P}$, then I is a model of $\mathcal{P}_H$ [4]. 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 $\mathcal{P} \models$ not(l), where l is a literal if every stable model satisfying $\mathcal{P}$ *does not* satisfy l.

Example 2 *Consider* $\mathcal{P} = \{(p, \neg q \leftarrow not(\neg r)), (r \leftarrow \neg s), (t \leftarrow p), (t \leftarrow \neg q)\}$. $\mathcal{P}$ *has two stable models* $\mathcal{I}_1 = \{p, t\}$ *and* $\mathcal{I}_2 = \{\neg q, t\}$, *so* $\mathcal{P} \models t$, *while* $\mathcal{P} \models not(r)$. *Note that the program* $\{a \leftarrow not(a)\}$ *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 $\bar{l} = \bar{p}(t_1, \ldots, t_n)$ if l is a negative literal $\neg p(t_1, \ldots, t_n)$. With $\bar{\mathcal{P}}$ we indicate the *positive form* of $\mathcal{P}$ obtained from $\mathcal{P}$ by replacing every literal occurring in $\mathcal{P}$ with its positive form. The definition of $\bar{X}$, where X is a set of extended literals is similar. Then, an interpretation $\mathcal{I}$ is a stable model of a program $\mathcal{P}$ iff $\bar{\mathcal{I}}$ is a stable model of $\bar{\mathcal{P}}$ [6].

Description Logic Programs. A *Description Logic Program* (DLP) is a pair $\mathcal{DP} = \langle \mathcal{T}, \mathcal{P} \rangle$, 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 β of a rule

[4] Note that $\mathcal{P}_H$ contains grounded extended literals

[5] The term 'description logic program' has already been used in [7], but with a slightly different meaning.

$\gamma \leftarrow \delta \in \mathcal{P}$ 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 $A(a) \leftarrow$, while an assertion $(a,b){:}R$ is represented by means of the rule $R(a,b) \leftarrow$. Note that we do not allow concept and role names that appear in $\mathcal{T}$ 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* for $\mathcal{DP} = \langle \mathcal{T}, \mathcal{P} \rangle$ is a pair $\mathcal{I} = (H, I)$, where $\mathcal{I}$ is an interpretation for $\mathcal{P}$ and $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is an interpretation for $\mathcal{T}$, where $\Delta^{\mathcal{I}} = H$, and for concept names A and roles names R, $A^{\mathcal{I}} = \{a \mid A(a) \in I\}$ and $R^{\mathcal{I}} = \{(a,b) \mid R(a,b) \in I\}$, respectively. An interpretation is a *model* of $\mathcal{DP} = \langle \mathcal{T}, \mathcal{P} \rangle$ iff it is a model of $\mathcal{T}$ and a stable model of $\mathcal{P}$. The definition of entailment is as usual.

Example 3 *Consider* $\mathcal{DP} = \langle \mathcal{T}, \mathcal{P} \rangle$*, with* $\mathcal{T} = \{A{:=}\forall R.\neg C, B{:=}\forall R.D, E{:=}C \sqcap D\}$ *and* $\mathcal{P} = \{(p(X) \leftarrow A(X)), (p(X) \leftarrow R(X,Y), E(Y), not(r(y))), (B(a) \leftarrow)\}$*. Then, by reasoning like in Example 1, it follows that* $\mathcal{DP} \models p(a)$ *. In fact,* $\mathcal{DP}$ *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 model, together with the rules and facts, 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 $\mathcal{SHIQ}^*$, 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 $\mathcal{DP} = \langle \mathcal{T}, \mathcal{P} \rangle$. The *closure*, $closure(\mathcal{DP})$, of $\mathcal{DP}$ is defined as follows. For every concept expression D and primitive role R in $\mathcal{DP}$ we have $\{D, R\} \subseteq closure(\mathcal{DP})$ and for every D in $closure(\mathcal{DP})$, we have one of the following: (i) if $D = \neg D'$, then $D' \in closure(\mathcal{DP})$; (ii) if $D = D' \sqcap D''$, then $\{D', D''\} \subseteq closure(\mathcal{DP})$; (iii) if $D = D' \sqcup D''$, then $\{D', D''\} \subseteq closure(\mathcal{DP})$; (iv) if $D = \exists R.D'$, then $\{R, D'\} \subseteq closure(\mathcal{DP})$; (v) if $D = \forall R.D'$, then $\{D', \exists R.\neg D'\} \subseteq closure(\mathcal{DP})$; and (vi) for all $D \in closure(\mathcal{DP})$, $\neg D \in closure(\mathcal{DP})$. Let $\Omega(\mathcal{DP})$ be the disjunctive

logic program, obtained from $\mathcal{DP}$, as follows.

$closure(\mathcal{DP})$	$\Omega(\mathcal{DP})$
concept name A	$A(X), \text{not}(A(X)) \leftarrow$
role name R	$R(X,Y), \text{not}(R(X,Y)) \leftarrow$ $\neg R(X,Y), \text{not}(\neg R(X,Y)) \leftarrow$
expressions D	
$D = \neg D'$	$\neg D'(X) \leftarrow \text{not}(D'(X))$
$D = D' \sqcap D''$	$(D' \sqcap D'')(X) \leftarrow D'(X), D''(X)$
$D = D' \sqcup D''$	$(D' \sqcup D'')(x) \leftarrow D'(x)$ $(D' \sqcup D'')(x) \leftarrow D''(x)$
$D = \exists R.D'$	$(\exists R.D')(X) \leftarrow R(X,Y), D'(Y)$
$D = \forall R.D'$	$(\forall R.D')(X) \leftarrow \neg(\exists R.\neg D')(X)$
$A{:}= C \in \mathcal{T}$	$\leftarrow A(X), \text{not}(C(X))$ $\leftarrow \text{not}(A(X)), C(X)$

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

Theorem 1 *An interpretation $\mathcal{I}$ of $\mathcal{DP} = \langle \mathcal{T}, \mathcal{P} \rangle$ is a model of $\mathcal{DP}$ iff I is a model of $\mathcal{DP}' = \langle \emptyset, \mathcal{P}_{\mathcal{DP}} \rangle$, 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 $a_0{:}\,\mu_0 \leftarrow a_1{:}\,\mu_1, \ldots, a_n{:}\,\lambda_n$, where the a_i are atoms and the λ_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 $\mathcal{ALC}$. Our uncertainty description logic is the one presented in [17], which we call here $\mu\mathcal{ALC}$. 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 *μinterpretation* is now a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is the *domain*, whereas $\cdot^{\mathcal{I}}$ 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 $\cdot^{\mathcal{I}}$ has to satisfy the following equations: for all $d \in \Delta^{\mathcal{I}}$, $\top^{\mathcal{I}}(d) = 1$,

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

A *μassertion* (denoted $\mu\alpha$) is an expression $\langle \alpha \succeq c \rangle$ or $\langle \alpha \preceq c' \rangle$, where α is an $\mathcal{ALC}$ assertion and $c, c' \in [0, 1]$. From a semantics point of view, a μassertion $\langle \alpha \preceq c \rangle$ constrains the certainty value of α to be less or equal to c (similarly for $\succeq$). An μinterpretation $\mathcal{I}$ *satisfies* $\langle a{:}C \succeq c \rangle$ (respectively $\langle (a, b){:}R \succeq c \rangle$) iff $C^{\mathcal{I}}(a^{\mathcal{I}}) \succeq c$ (respectively $R^{\mathcal{I}}(a^{\mathcal{I}}, b^{\mathcal{I}}) \succeq c$). Similarly for $\preceq$. Note that, e.g. $\langle a{:}\neg C \succeq c \rangle$ and $\langle a{:}C \preceq 1 - c \rangle$ are satisfied by the same set of μinterpretations. Concerning terminological axioms, an μinterpretation $\mathcal{I}$ *satisfies* $A \sqsubseteq C$ iff $\forall d \in \Delta^{\mathcal{I}}, A^{\mathcal{I}}(d) \leq C^{\mathcal{I}}(d)$, while $\mathcal{I}$ *satisfies* $A{:=}C$ iff $\forall d \in \Delta^{\mathcal{I}}, A^{\mathcal{I}}(d) = C^{\mathcal{I}}(d)$. A *$\mu$Knowledge Base* ($\mu$KB) is pair $\mu\mathcal{K} = \langle \mathcal{T}, \mu\mathcal{A} \rangle$, where $\mathcal{T}$ and $\mu\mathcal{A}$ are finite sets of terminological axioms and μassertions, respectively. The notions of *satisfiability* (*model*) of a μKB and that of *entailment* are as usual. Finally, given $\mu\mathcal{K}$ and an assertion α, it is of interest to compute α's best lower and upper certainty value bounds. The *greatest lower bound* of α w.r.t. $\mu\mathcal{K}$ (denoted $glb(\mu\mathcal{K}, \alpha)$) is $\sup\{c : \mu\mathcal{K} \models \langle \alpha \succeq c \rangle\}$, while the *least upper bound* of α with respect to $\mu\mathcal{K}$ (denoted $lub(\mu\mathcal{K}, \alpha)$) is $\inf\{c : \mu\mathcal{K} \models \langle \alpha \preceq c \rangle\}$ ($\sup \emptyset = 0, \inf \emptyset = 1$). Determining the *lub* and the *glb* is called the *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 $\mu\mathcal{K} = \langle \mathcal{T}, \mu\mathcal{A} \rangle$, with $\mathcal{T} = \{A{:=}\forall R.\neg B\}$, $\mu\mathcal{A} = \{\langle a{:}\forall R.C \succeq 0.7 \rangle\}$. For $\alpha = a{:}A \sqcup \exists R.(B \sqcap C)$, $glb(\mu\mathcal{K}, \alpha) = 0.5$ and $lub(\mu\mathcal{K}, \alpha) = 1$ 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 *annotation function* of arity n is a total and computable function [6] $f{:}([0, 1])^n \to [0, 1]$. 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*, ν, 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 ν_i are annotation items. An *annotation term*, λ, is of the form $[\nu, \nu']$, where ν and ν' are annotation items. Annotation terms are supposed to denote subintervals of $[0, 1]$.

Let l be a literal and λ an annotation term. A *μliteral*, denoted μl, is of the form $l{:}\lambda$. The intended meaning is that "the certainty of l lies in the interval λ". An *extended μliteral* is of the form $\mathrm{not}(\mu l)$, where μl is a μliteral. The intended meaning of $\mathrm{not}(l{:}\lambda)$ is that "it is not provable that the certainty of l lies in the interval λ". A *μdisjunctive logic program*, $\mu\mathcal{P}$, is a finite set of *μrules* of the form $\gamma \leftarrow \delta$, where γ and δ are finite sets of extended μliterals. In order to avoid straightforward repetition, if not stated otherwise, definitions related to μdisjunctive logic programs, parallels those for disjunctive logic programs. Furthermore, in grounding a μliteral $l{:}\lambda$, we assume that the annotation term λ 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 μprogram $\mu\mathcal{P}$ may contain an infinite number of rules due to the grounding of annotation terms. For a grounded μprogram $\mu\mathcal{P}$, $\mathcal{B}_{\mu\mathcal{P}}$ 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 *μinterpretation* I of a grounded μprogram $\mu\mathcal{P}$ is any (possibly partial) function $I\colon \mathcal{B}_{\mu\mathcal{P}} \rightarrow [0,1]$ (*some ground atoms may be left unspecified*). The set of defined atoms in I is denoted $def(I)$. In the following, whenever we write $I(a)$, we assume that $a \in def(I)$. We extend I to literals $l = \neg a$ in the obvious way: $I(l) = 1 - I(a)$. A μinterpretation I *satisfies* a ground μliteral $l\colon \lambda$ iff $I(l) \in \lambda$. Note that we can always assume that μliterals are positive, by replacing $\neg a\colon [\nu_1, \nu_2]$ with $a\colon [\neg\nu_2, \neg\nu_1]$. Like for disjunctive logic programs, we say that I *satisfies* an extended μliteral not$(l\colon \lambda)$ iff I does not satisfy $l\colon \lambda$, i.e. $I(l) \notin \lambda$. An μinterpretation I of a grounded program $\mu\mathcal{P}$ without naf *satisfies* a rule $\gamma \leftarrow \delta$ iff if I satisfies every μliteral in δ then I satisfies some μliteral in α. An μinterpretation I is a *μmodel* of program $\mu\mathcal{P}$ without naf iff it satisfies every rule in $\mu\mathcal{P}$.

Example 5 *Consider the grounded μprogram $\mu\mathcal{P}$ without naf,* $\{(c\colon [0.1, 0.3] \leftarrow),$ $(a\colon [0.2, 0.7], b\colon [0.3, 0.6] \leftarrow)\}$. *Let us consider the following two partial functions $\mathscr{I}_1$ and $\mathscr{I}_2$, assigning to atoms intervals and defined as follows: $\mathscr{I}_1(a) = [0.2, 0.7], \mathscr{I}_1(c) = [0.1, 0.3]$, while $\mathscr{I}_2(b) = [0.3, 0.6], \mathscr{I}_2(c) = [0.1, 0.3]$. It is easily verified that for each μmodel I of $\mu\mathcal{P}$, we have that for some $i = 1, 2$, $def(\mathscr{I}_i) \subseteq def(I)$ and for all ground atoms $p \in def(\mathscr{I}_i), I(p) \in \mathscr{I}_i(p)$. Essentially, $\mathscr{I}_1$ and $\mathscr{I}_2$ may be seen as a partitioning of the μmodels, such that $\mathscr{I}_i$ 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 $\mathcal{C}[0,1]$ be the set of all closed sub-intervals of $[0,1]$. We also add $\emptyset$ to $\mathcal{C}[0,1]$, called the empty interval. We will use it to manage inconsistencies among intervals. For two intervals σ_1 and σ_1 in $\mathcal{C}[0,1]$, we define $\sigma_1 \preceq_p \sigma_2$ iff $\sigma_2 \subseteq \sigma_1$. Similarly, $\sigma_1 \prec_p \sigma_2$ 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* $\mathscr{I}$ of a grounded program $\mu\mathcal{P}$ without naf, is a (possibly partial) function $\mathscr{I}\colon \mathcal{B}_{\mu\mathcal{P}} \rightarrow \mathcal{C}[0,1]$. An interval interpretation $\mathscr{I}$ is a representative of a whole family of μinterpretations I: we write $I \in \mathscr{I}$ iff $def(I) = def(\mathscr{I})$ and for all $a \in def(\mathscr{I})$, $I(a) \in \mathscr{I}(a)$ (see Example 5). We extend interval interpretations $\mathscr{I}$ to ground literals $l = \neg a$ as usual: $\mathscr{I}(l) = \neg\mathscr{I}(a)$. For interval interpretations $\mathscr{I}_1$ and $\mathscr{I}_2$, $\mathscr{I}_1 \preceq_p \mathscr{I}_2$ iff $def(\mathscr{I}_2) \subseteq def(\mathscr{I}_1)$ and for all $a \in def(\mathscr{I}_2), \mathscr{I}_1(a) \preceq_p \mathscr{I}_2(a)$. $\mathscr{I}_1 \prec_p \mathscr{I}_2$ iff $\mathscr{I}_1 \preceq_p \mathscr{I}_2$ and for some $a \in def(\mathscr{I}_2), \mathscr{I}_1(a) \prec_p \mathscr{I}_2(a)$. The $\preceq$-least interval interpretation, $\mathscr{I}_\perp$, assigns to all ground atoms in $\mathcal{B}_{\mu\mathcal{P}}$ the interval $[0,1]$, meaning essentially that the certainty of the atoms is *unknown*. An interval interpretation $\mathscr{I}$ *satisfies* a ground μliteral $l\colon \lambda$ iff $\lambda \preceq \mathscr{I}(l)$. An interval interpretation $\mathscr{I}$ of a grounded program $\mu\mathcal{P}$ without naf *satisfies* a rule $\gamma \leftarrow \delta$ iff if $\mathscr{I}$ satisfies every μliteral in δ then $\mathscr{I}$ satisfies some μliteral in α. An interval interpretation $\mathscr{I}$ is an *interval model* of program $\mu\mathcal{P}$ without naf iff it satisfies every rule in $\mu\mathcal{P}$. Furthermore, $\mathscr{I}$ is *minimal* as well iff there is no interval model $\mathscr{I}' \prec \mathscr{I}$ of $\mu\mathcal{P}$. For instance, in Example 5, $\mathscr{I}_1$ and $\mathscr{I}_2$ 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 naf, then there is an unique minimal interval model for $\mu\mathcal{P}$, which is the $\preceq$-least fixed-point, $lfp(T_{\mu\mathcal{P}})$, of the following $T_{\mu\mathcal{P}}$ monotone operator: for all $a \in \mathcal{B}_{\mu\mathcal{P}}$,

$T_{\mu\mathcal{P}}(\mathscr{I})(a) = \bigcap\{\lambda \mid a{:}\,\lambda \leftarrow \beta \in \mu\mathcal{P}, \mathscr{I} \text{ satisfies each } \mu\text{literal in } \beta\}$. Note that $T_{\mu\mathcal{P}}$ is not continuous. In fact the grounded μprogram without naf, containing all ground instances of the rules $(a{:}\,[0,1] \leftarrow)$, $(a{:}\,[\frac{y+1}{2}, y] \leftarrow a{:}\,[x,y])$ and $(b{:}\,[1,1] \leftarrow a{:}\,[1,1])$ has unique minimal interval model $\mathscr{I}(a) = \mathscr{I}(b) = [1,1]$, which is obtained after $\omega + 1$ $T_{\mu\mathcal{P}}$ iterations over $\mathscr{I}_\perp$, where ω is the first limit ordinal. Note also that the least and unique interval model $\mathscr{I}$ of the μprogram $\{(a{:}\,[0,0] \leftarrow), (a{:}\,[1,1] \leftarrow)\}$ assigns to a the empty set $\emptyset$, which indicates that there is no μmodel of it. We may even use this property to manage inconsistencies of this kind, by allowing interpretations to map ground atoms into a new symbol, indicating that the μprogram is inconsistent on that atom. We will not investigate this issue further in this paper.

Given a grounded μprogram (possibly with naf) $\mu\mathcal{P}$ and an μinterpretation I for $\mu\mathcal{P}$, the Gelfond-Lifschitz transformation, is the grounded positive μprogram $\mu\mathcal{P}^{\mathcal{I}}$, obtained by deleting in $\mu\mathcal{P}$, (i) each rule that has not(μl) in its body and I satisfies μl; (ii) each rule that has not(μl) in its head and I does not satisfy μl; and (iii) all not(μl) in the bodies and heads of the remaining rules. Finally, an *μinterpretation* of a μprogram $\mathcal{P}$ (possibly not grounded) is a pair $\mathcal{I} = (H, I)$, such that I is an μinterpretation of the grounded program $\mu\mathcal{P}_H$. An *μinterpretation* $\mathcal{I} = (H, I)$ of a μprogram $\mu\mathcal{P}$ is a *stable μmodel* of $\mu\mathcal{P}$ iff $I \in \mathscr{I}$ for a minimal interval model $\mathscr{I}$ of $\mu\mathcal{P}_H^I$. Finally, we say that a program $\mu\mathcal{P}$ *entails* a ground extended μliteral μl, denoted $\mu\mathcal{P} \models \mu l$ iff every stable μmodel of $\mu\mathcal{P}$ satisfies μl. For instance, in Example 5, we have that any stable μmodel I of $\mu\mathcal{P}$ is such that $I \in \mathscr{I}_1$ or $I \in \mathscr{I}_2$.

Example 6 *Consider the following μprograms $\mu\mathcal{P}_1, \mu\mathcal{P}_2, , \mu\mathcal{P}_3$ and $\mu\mathcal{P}_4$, where $\mu\mathcal{P}_1 = \{r_1, r_2\}, \mu\mathcal{P}_2 = \{r_1, r_3\}$, $\mu\mathcal{P}_3 = \{r_1, r_4\}$, $\mu\mathcal{P}_4 = \{r_1, r_5\}$ and the rules r_i are $(r_1 : a{:}\,[0.6, 0.8] \leftarrow)$, $(r_2 : b{:}\,[0.4, 0.5] \leftarrow not(a{:}\,[0.2, 0.3]))$, $(r_3 : b{:}\,[0.4, 0.5] \leftarrow not(a{:}\,[0.2, 0.7]))$, $(r_4 : b{:}\,[x, y] \leftarrow a{:}\,[x, y])$ and $(r_5 : b{:}\,[x, y] \leftarrow not(a{:}\,[x, y]))$. It can be verified that for any stable μmodel I of $\mu\mathcal{P}_i$ we have that (i) for $\mu\mathcal{P}_1$, $I(a) \in [0.6, 0.8]$ and $I(b) \in [0.4, 0.5]$. Therefore, $\mu\mathcal{P}_1 \models b{:}\,[0.4, 0.5]$; (ii) for $\mu\mathcal{P}_2$, $I(a) \in [0.6, 0.8]$ and if $I(a) \in (0.7, 0.8]$ then $I(b) \in [0.4, 0.5]$ else I is undefined on b. Therefore, $\mu\mathcal{P}_2 \not\models b{:}\,[c, c']$, for any $c, c' \in [0, 1]$. But, $\mu\mathcal{P}_2 \models not(b{:}\,[c, c'])$ if $[c, c'] \cap [0.4, 0.5] = \emptyset$; and (iii) for $\mu\mathcal{P}_3$, $I(a) \in [0.6, 0.8]$ and $I(b) \in [0.6, 0.8]$. Therefore, $\mu\mathcal{P}_3 \models b{:}\,[0.6, 0.8]$. For $\mu\mathcal{P}_4$, first note that for any I satisfying r_1, $I(a) \in [0.6, 0.8]$ holds. Therefore, $\mu\mathcal{P}_4^I = \{r_1\} \cup \{(b{:}\,[c, c'] \leftarrow) | I(a) \notin [c, c'], c \leq c' \in [0, 1]\}$, whose least interval model $\mathscr{I}$ is such that $\mathscr{I}(a) = [0.4, 0.5]$ and $\mathscr{I}(b) = \emptyset$. So, $\mu\mathcal{P}_4$ has no stable model.*

Uncertainty in Description Logic Programs. A *μDescription Logic Program* (μDLP), denoted $\mu\mathcal{D}P$, is a pair $\langle \mathcal{T}, \mu\mathcal{P} \rangle$, where $\mathcal{T}$ is a $\mu\mathcal{ALC}$ terminology and $\mu\mathcal{P}$ is a μprogram, where roles names and concept names appearing in $\mathcal{T}$ may appear also in the body of the rules, and μassertions of the form $\langle a{:}A \succeq c \rangle$, where A is a concept name, are represented by means of the rule $A(a){:}\,[c, 1] \leftarrow$. Similarly for $\langle a{:}A \preceq c \rangle$, $\langle (a, b){:}R \succeq c \rangle$ and $\langle (a, b){:}R \preceq c \rangle$.

An interpretation for a μDLP $\mu\mathcal{D}P$ is a pair $\mathcal{I} = (H, I)$, where $\mathcal{I}$ is a μinterpretation for the μprogram $\mu\mathcal{P}$ and $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is a μinterpretation for the terminology $\mathcal{T}$, where $\Delta^{\mathcal{I}} = H$, and for concept names A and roles names R, $A^{\mathcal{I}}(a) = I(A(a))$ and $R^{\mathcal{I}}(a) = I(R(a, b))$, respectively. An interpretation for a μDLP $\mu\mathcal{D}P$ is a *μmodel* of a μDLP $\mu\mathcal{D}P = \langle \mathcal{T}, \mu\mathcal{P} \rangle$ iff I is a μmodel of $\mathcal{T}$ and a stable μmodel of $\mu\mathcal{P}$. Entailment is defined as usual.

Example 7 *Let us consider the DLP of Example 3, but extended with uncertainty as follows. $\mu\mathcal{D}P = \langle \mathcal{T}, \mathcal{P} \rangle$, where $T = \{A{:}= \forall R.\neg C, B{:}= \forall R.D, E{:}= C \sqcap D\}$ and*

$\mu\mathcal{P} = \{(B(a)\colon[0.7,1] \leftarrow), (p(X)\colon[x,y] \leftarrow A(X)\colon[x,y]),\ (p(X)\colon[x,y] \leftarrow R(X,Y)\colon[x,y],\ E(Y)\colon[x,y], not(r(Y)\colon[0,1,0.2]))\}$. *Consider the atom* $p(a)$. *Then, by reasoning by cases like in Example 3, it follows that* $\mu\mathcal{DP} \models p(a)\colon[0.5,1]$. *In fact, the* μ*models* $\mathcal{I} = (H, I)$ *of* $\mu\mathcal{DP}$ *are such that* $H = \{a\}$ *and either* $def(I) = \{B(a), A(a), p(a)\}$ *or* $def(I) = \{B(a), R(a,a), C(a), D(a), E(a), p(a)\}$. *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* μ*model* $\mathcal{I} = (H, I)$ *of* $\mu\mathcal{DP}$, *we have* $I(p(a)) \geq 0.5$.

Reasoning in μDLP. Reasoning in μDLP is not an easy task. From computational point of view, several sources of complexity come to play a significant role: the computational complexity of the description logic component, the one of the logic programming component and the one from the literal annotation part. Due to the expressive power of μDLP 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 $T_{\mu\mathcal{P}}$ is indicating a similar result for normal μprograms without naf. In the general case, an extension of the existential entail algorithm [13] to the our uncertainty 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 'μtranslate' a concept $D = \exists R.D'$ into a rule [7] $(\exists R.D')(X)\colon[\min(x,x'), \min(y,y')] \leftarrow R(X,Y)\colon[x,y], D'(Y)\colon[x',y']$ seems to be more promising. Of course, we have to pay some attention with 'infinity'. On the non-annotation terms part, we have to restrict our attention to so-called *conceptual logic programs* [8], to ensure that the stable models (set of ground atoms) is finite. Furthermore, we have to ensure that grounding a μprogram with certainty values is finite as well.

Theorem 2 *Consider a* μ*DLP,* $\mu\mathcal{DP} = \langle\mathcal{T}, \mu\mathcal{P}\rangle$, *where the annotation functions are either* $\max, \min$ *or* $1-$. *Consider its '*μ*translation' into a* μ*DLP without terminology,* $\mu\mathcal{DP}' = \langle\emptyset, \mu\mathcal{P}_{\mu\mathcal{DP}}\rangle$, *such that the program* $\mathcal{P}_{\mathcal{DP}}$ *obtained from* $\mu\mathcal{P}_{\mu\mathcal{DP}}$ *by removing annotation terms is a conceptual logic program [8]. Then the entailment problem* (i) *is decidable* (ii) *by means of a standard disjunctive logic program reasoner.*

Proof.[Sketch] The key point is that the annotation functions are restricted in the form so to guarantee that the set of certainty values to be considered is finite. Like in [17], we can restrict our attention to annotation terms instantiations, where the certainty values are taken from the set $N^{\mu\mathcal{DP}} = \{0, 0.5, 1\} \cup \{c \mid c \text{ occurs in } \mu\mathcal{DP}\} \cup \{1-c : c \text{ occurs in } \mu\mathcal{DP}\}$. Therefore, the set of grounded rules is finite and we can enumerate all models of the μDLP and check for entailment, which completes part (i). For part (ii), we observe that any μatom with grounded annotation term $p(\ldots)\colon[c,c']$, may be seen as a new atom without annotation term, $(p[c,c'])(\ldots)$, by replacing p with a new symbol $p[c,c']$. The effect of this replacement is that we obtain a disjunctive logic program $\mathcal{P}'$. Finally, we have to encode the relationships between the new atoms, reflecting the semantics of intervals: for any interval $\sigma, \sigma_1, \sigma_2$ occurring in $\mathcal{P}'$, we sadd the following rules to $\mathcal{P}'$: (i) $(p\sigma_1)(X_1,\ldots,X_n) \leftarrow (p\sigma_2)(X_1,\ldots,X_n)$ if $\sigma_2 \subseteq \sigma_1$; (ii) $(p\sigma)(X_1,\ldots,X_n) \leftarrow (p\sigma_1)(X_1,\ldots,X_n),\ (p\sigma_2)(X_1,\ldots,X_n)$ if $\sigma = \sigma_1 \cap \sigma_2$; (iii) $(p\sigma)(X_1,\ldots,X_n),\ \text{not}((p\sigma)(X_1,\ldots,X_n)) \leftarrow$; (iv) $(\tilde{p}\sigma)(X_1,\ldots,X_n) \leftarrow$

[7] We omit the complete translation of all expressions as it parallels the one presented early.

not$((p\sigma)(X_1,\ldots,X_n))$; and $(v) \leftarrow (p\sigma_1)(X_1,\ldots,X_n), (\tilde{p}\sigma_2)(X_1,\ldots,X_n)$ if $\sigma_1 \subseteq \sigma_2$. $\tilde{p}$ is a new symbol associated to p. $\tilde{p}\sigma(\cdot)$ 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 $\mu\mathcal{D}P_2$ 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 $\mu\mathcal{D}P_2$ 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 $N^{\mu\mathcal{D}P}$ generates a finite set, or where the μDLP does not contain recursive rules.

4 Conclusions

We integrated the management of uncertainty into a highly expressive family of representation languages, called μDLP, resulting from the combination of description logics and disjunctive logic programs. We defined syntax, semantics and discussed some reasoning issues related to μDLP. 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 μDLPs or to meaningful fragments of them.

References

[1] A. Aslam, Javed and Mark Montague. Models of metasearch. In *Proceedings of the 24rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (ACM SIGIR-01)*, pages 276–284, New Orleans, USA, 2001.

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

[3] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *The Scientific American*, 284(5):34–43, 2001.

[4] Tina Dell'Armi, Wolfgang Faber, Giuseppe Ielpa, Christoph Koch, Nicola Leone, Simona Perri, and Gerald Pfeifer. System description: DLV. In *6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-01)*, number 2173 in Lecture Notes in Artificial Intelligence, pages 409–412, 2001.

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

[6] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3/4):365–386, 1991.

[7] Benjamin Grosof and Ian Horrocks. Description logics programs: Combining logic programs with description logics. In *The Twelfth International World Wide Web Conference (WWW-03)*, pages –, Budapest, Hungary, 2003.

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

[9] Ian Horrocks and Peter F. Patel-Schneider. Three theses of representation in the semantic web. In *The Twelfth International World Wide Web Conference (WWW-03)*, pages –, Budapest, Hungary, 2003.

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

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

[12] Maurizio Lenzerini. Data integration: a theoretical perspective. In *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS-02)*, pages 233–246. ACM Press, 2002.

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

[14] Vladimir Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1-2):39–54, 2002.

[15] Raymond Ng and V.S. Subrahmanian. Stable model semantics for probabilistic deductive databases. In Zbigniew W. Ras and Maria Zemenkova, editors, *Proc. of the 6th Int. Sym. on Methodologies for Intelligent Systems (ISMIS-91)*, number 542 in Lecture Notes in Artificial Intelligence, pages 163–171. Springer-Verlag, 1991.

[16] I. Niemelä and P. Simons. Smodels - an implementation of the stable model and well-founded semantics for normal logic programs. In *Proceedings of the 4th International Conference on Logic Programming and Non-Monotonic Reasoning*, pages 420–429. Springer-Verlag, 1997.

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