An Approach to Handling Inconsistent Ontology Definitions based on the Translation of Description Logics into Defeasible Logic Programming

Sergio A. Gómez

and

Carlos I. Chesñevar

and

Guillermo R. Simari

Laboratorio de Investigación y Desarrollo en Inteligencia Artificial (LIDIA)*
Depto. Cs. e Ing. de la Computación – Universidad Nacional del Sur
Alem 1253 (8000) Bahía Blanca - ARGENTINA –
Tel/Fax: (+54) 291-459 5135/5136 – E-mail: {sag, cic, grs}@cs.uns.edu.ar

Abstract

The *Semantic Web* is a future vision of the web where stored information has exact meaning, thus enabling computers to understand and reason on the basis of such information. Assigning semantics to web resources is addressed by means of *ontology definitions* which are meant to be written in an *ontology description language* such as OWL-DL that is based on so-called Description Logics (DL). Although ontology definitions expressed in DL can be processed with existing DL reasoners, such DL reasoners are incapable of dealing with *inconsistent* ontology definitions.

Previous research has determined that a subset of DL can be effectively translated into an equivalent subset of logic programming. We propose a method for dealing with inconsistent ontology definitions in the Semantic Web. Our proposal involves mapping DL ontologies into equivalent DeLP programs. That is, given an OWL-DL ontology O_{Owl}, an equivalent DL ontology O_{DL} can be obtained. Provided O_{DL} satisfies certain restrictions, it can be translated into an equivalent DeLP program O_{DeLP}. Therefore, given a query Q w.r.t. O_{Owl}, a dialectical process will be performed to determine if Q is warranted w.r.t. O_{DeLP}.

Keywords: Semantic web, defeasible argumentation, Description Logics, Defeasible Logic Programming, inconsistent ontology handling

1 INTRODUCTION AND MOTIVATIONS

The *Semantic Web* [3] is a future vision of the web where stored information has exact meaning, thus enabling computers to understand and reason on the basis of such information. Assigning semantics to web resources is addressed by means of *ontology definitions*. The term ontology stands for a

*LIDIA is a member of IICyTI (Instituto de Investigación en Ciencia y Tecnología Informática).

specification of a conceptualization, that is to say a description of the concepts and relationships that can exist for an agent or a community of agents.

As proposed by the World Wide Web Consortium (W3C)[1], ontology definitions are meant to be written in an *ontology description language* such as OWL [14], whose subset known as OWL-DL is based on so-called Description Logics (DL) [1]. Although ontology definitions expressed in DL can be processed with existing DL reasoners (such as RACER [12]), such DL reasoners are incapable of dealing with *inconsistent* ontology definitions. In particular, when a DL reasoner is presented with an inconsistent ontology definition, it will be unable to extract any useful consequences from it.

However, previous research [11] has determined that a subset of DL can be effectively translated into an equivalent subset of Horn logic. In particular, ***DeLP*** is an argumentative framework based on logic programming that is capable of dealing with possibly inconsistent knowledge bases codified as a set of Horn-like clauses [8]. When presented with a query Q w.r.t. to a KB $\mathcal{P}$, DeLP performs a *dialectical* process in which all *arguments* in favor and against Q are considered. In such a process, arguments regarded as ultimately *undefeated* will be considered *warranted*.

In this article, we propose a method for dealing with inconsistent ontology definitions in the Semantic Web. Our proposal involves mapping DL ontologies into an equivalent DeLP program. That is, given an OWL-DL ontology O_{Owl}, an equivalent DL ontology O_{DL} can be obtained. Provided O_{DL} satisfies certain restrictions, it can be translated into an equivalent DeLP program O_{DeLP}. Then given a query Q, a dialectical process as explained above will be performed to determine if Q is warranted w.r.t. O_{DeLP}.

The rest of this paper is structured as follows. Section 2 introduces the fundamentals of Description Logics. Section 3 briefly explains the Defeasible Logic Programming formalism. Section 4 introduces how the mapping from DL to DeLP is performed. Section 5 explains how inconsistent ontology definitions are handled within DeLP. Section 6 discusses related work. Finally Section 7 concludes.

2 DESCRIPTION LOGICS

Description Logics (DL) are a well-known family of knowledge representation formalisms [1]. They are based on the notions of *concepts* (unary predicates, classes) and *roles* (binary relations), and are mainly characterized by constructors that allow complex concepts and roles to be built from atomic ones. The expressive power of a DL system is determined by the constructs available for building concept descriptions, and by the way these descriptions can be used in the terminological (*Tbox*) and assertional (*Abox*) components of the system.

We now describe the basic language for building DL expressions. Let C and D stand for concepts and R for a role name. Concept descriptions are built from concept names using the constructors conjunction $(C \sqcap D)$, disjunction $(C \sqcup D)$, negation $(\neg C)$, existencial restriction $(\exists R.C)$, and value restriction $(\forall R.C)$. To define the semantics of concept descriptions, concepts are interpreted as subsets of a domain of interest, and roles as binary relations over this domain. An interpretation I consists of a non-empty set Δ^I (the domain of I) and a function $\cdot^I$ (the interpretation function of I) which maps every concept name A to a subset A^I of Δ^I, and every role name to R to a subset R^I of $\Delta^I \times \Delta^I$. The interpretation function is extended to arbitrary concept descriptions as follows: $(\neg C)^I = \Delta^I \backslash C^I$; $(C \sqcup D)^I = C^I \cup D^I$; $(C \sqcap D)^I = C^I \cap D^I$; $(\exists R.C)^I = \{x | \exists y \text{ s.t. } (x, y) \in R^I \text{ and } y \in C^I\}$, and $(\forall R.C)^I = \{x | \forall y, (x, y) \in R^I \text{ implies } y \in C^I\}$. Besides, the expressions $\top$ and $\bot$ are shorthands for $C \sqcup \neg C$ and $C \sqcap \neg C$, resp. Further extensions to the basic DL are possible including inverse and

[1] www.w3c.org

$$Tbox = \left\{ \begin{array}{ll} (1) & bird \sqsubseteq animal; \\ (2) & bird \sqsubseteq fly; \\ (3) & eagle \sqsubseteq bird; \\ (4) & penguin \sqsubseteq bird; \\ (5) & penguin \sqsubseteq \neg fly; \\ (6) & penguin \sqcap \exists isOperatedBy.geneticSurgeon \sqsubseteq geneticallyAlteredPenguin; \\ (7) & geneticallyAlteredPenguin \sqsubseteq fly; \\ (8) & eagle \sqcap hasBrokenWing \sqsubseteq \neg fly; \\ (9) & isOperatedBy \equiv operates^{-}; \\ (10) & geneticSurgeon \equiv genetist \sqcap surgeon \end{array} \right\}$$

$$Abox = \left\{ \begin{array}{ll} (11) & penguin(opus); \\ (12) & eagle(avenger); \\ (13) & hasBrokenWing(avenger); \\ (14) & genetist(frankenstein); \\ (15) & surgeon(frankenstein); \\ (16) & operates(frankenstein, opus) \end{array} \right\}$$

Figure 1: Knowledge base $\mathcal{KB}$ about flying animals expressed in Description Logics

transitive roles noted as P^{-} and P^{+}, resp.

A DL knowledge base (KB) consists of two finite and mutually disjoint sets: $Tbox$ which introduces the *terminology* and $Abox$ which contains facts about particular objects in the application domain. Tbox statements have the form $C \sqsubseteq D$ (*inclusions*) and $C \equiv D$ (*equalities*), where C and D are (possibly complex) concept descriptions. The semantics of Tbox statements is as follows. An interpretation I satisfies $C \sqsubseteq D$ iff $C^I \subseteq D^I$, I satisfies $C \equiv D$ iff $C^I = D^I$. Objects in the Abox are referred to by a finite number of *individual names* and these names may be used in two types of assertional statements: *concept assertions* of the type $C(a)$ and *role assertions* of the type $R(a, b)$, where C is a concept description, R is a role name, and a and b are individual names. An interpretation I *satisfies* the assertion $C(a)$ iff $a^I \in C^I$, and it *satisfies* $R(a, b)$ iff $(a^I, b^I) \in R^I$. An interpretation I is a *model* of a DL (Tbox or Abox) statement ϕ iff it satisfies the statement, and is a model of a DL knowledge base $\mathcal{KB}$ iff it satisfies every statement in $\mathcal{KB}$. A DL knowledge base $\mathcal{KB}$ *entails* a DL statement ϕ, written as $\mathcal{KB} \models \phi$, iff every model of $\mathcal{KB}$ is a model of ϕ.

Example 1 *Consider the ontology $\mathcal{KB} = (Tbox, Abox)$ in Figure 1 about flying animals. Notice that although $Tbox$ and $Abox$ are sets of sentences, its elements are enumerated for clarity. Sentence (1) in $Tbox$ says that every bird is an animal, sentence (2) says that every bird flies. Sentences (3) and (4) say that eagles and penguins are birds. Sentences (5), (6) and (7) say that penguins do not fly unless they are genetically-altered, where a genetically-altered penguin is a penguin that has been operated by a genetic surgeon. Sentence (8) establishes that eagles with a broken wing are no longer able to fly. Sentence (9) defines relation $isOperatedBy$ as the inverse of relation $operates$. Finally, sentence (10) defines a genetic surgeon as a genetist who is also a surgeon.*

Sentence (11) in $Abox$ expresses that Opus is a penguin. Sentences (12) and (13) establish that Avenger is an eagle who has a broken wing. Sentences (14) and (15) say that Frankenstein is both a genetist and a surgeon, resp. Finally, sentence (16) establishes that Frankenstein has operated Opus.

3 DEFEASIBLE LOGIC PROGRAMMING

Defeasible Logic Programming (DeLP) [8] provides a language for knowledge representation and reasoning that uses *defeasible argumentation* [6, 15, 16] to decide between contradictory conclusions

through a *dialectical analysis*. Codifying knowledge by means of a DeLP program provides a good trade-off between expressivity and implementability. Recent research has shown that DeLP provides a suitable framework for building real-world applications (*e.g.*, clustering algorithms [9], intelligent web search [5] and intelligent web forms [10]) that deal with incomplete and potentially contradictory information. In a defeasible logic program $\mathcal{P} = (\Pi, \Delta)$, a set Δ of defeasible rules $P \prec Q_1, \ldots, Q_n$, and a set Π of strict rules $P \leftarrow Q_1, \ldots, Q_n$ can be distinguished. An *argument* $\langle \mathcal{A}, H \rangle$ is a minimal non-contradictory set of ground defeasible clauses $\mathcal{A}$ of Δ that allows to derive a ground literal H possibly using ground rules of Π. Since arguments may be in conflict (concept captured in terms of a logical contradiction), an attack relationship between arguments can be defined. A criterion is usually defined to decide between two conflicting arguments. If the attacking argument is strictly preferred over the attacked one, then it is called a *proper defeater*. If no comparison is possible, or both arguments are equi-preferred, the attacking argument is called a *blocking defeater*. In order to determine whether a given argument $\mathcal{A}$ is ultimately undefeated (or *warranted*), a dialectical process is recursively carried out, where defeaters for $\mathcal{A}$, defeaters for these defeaters, and so on, are taken into account. Given a DeLP program $\mathcal{P}$ and a query H, the final answer to H w.r.t. $\mathcal{P}$ takes such dialectical analysis into account. The answer to a query can be: *yes*, *no*, *undecided*, or *unknown*. For an example of DeLP, see Examples 2 and 3 in Sections 4 and 5, resp.

4 TRANSLATING FROM DESCRIPTION LOGICS TO DEFEASIBLE LOGIC PROGRAMMING FOR REASONING WITH INDIVIDUALS

As explained above DL ontologies can be inconsistent. Notably DL reasoners such as RACER [12] are incapable of dealing with such situations. When presented with such inconsistent ontologies, RACER will issue an error message and will stop further processing.

Grosof *et al.* [11] have identified a subset of DL languages that can be effectively mapped into a Horn-clause logics. Accordingly, our work is based on such research by adapting it to the DeLP framework. We therefore propose translating a DL ontology $\mathcal{KB} = (Tbox, Abox)$ into an equivalent DeLP program $\mathcal{P} = (\Pi, \Delta)$ by means of a mapping $\mathcal{T}$ such that $\mathcal{P} = \mathcal{T}(\mathcal{KB})$. Intuitively Π in $\mathcal{P}$ will correspond to $Abox$ in $\mathcal{KB}$ while Δ will correspond to $Tbox$ in $\mathcal{KB}$. In the rest of this section, we will explain how to achieve the translation of DL KB into equivalent DeLP programs exemplifying the process with the DL KB described in Example 1. Moreover, defeasible rules of the form "$H \prec B_1, \ldots, B_n$" will be written for clarity as "$H \prec B_1 \wedge \ldots \wedge B_n$".

4.1 Translating Statements

For mantaining backward compatibility with previous knowledge representation languages, part of OWL-DL constructs are based on RDF Schema (RDFS). RDFS provides a subset of the DL statements (subclass, subproperty, range, and domain statements), which in a DL setting are called *Tbox* axioms, and asserted class-instance (type) and instance-property-instance relationships (which in a DL setting are called *Abox* axioms). A DL inclusion axiom corresponds to a first-order logic (FOL) implication[2]. Then class and property axioms maps to DeLP as: Class C is subclass of class D (noted as $C \sqsubseteq D$) maps to "$D(X) \prec C(X)$" and property Q is a subproperty of property P (noted as $Q \sqsubseteq P$) maps to "$P(X, Y) \prec Q(X, Y)$" where X and Y are variable names. The range and domain statements map as follows: the range of property P is class C (noted as $\top \sqsubseteq \forall P.C$) maps to "$C(Y) \prec P(X, Y)$" and the domain of property P is in class C (noted as $\top \sqsubseteq \forall P.C$) maps

[2]In particular the DL axiom $C \sqsubseteq D$ corresponds to the FOL formula $(\forall x)(C(x) \rightarrow D(x))$.

to "$C(Y) \prec P(Y, X)$". Asserted class-instance (type) and instance-property-instance relationships, which correspond to DL axioms of the form $C(a)$ and $P(a, b)$ resp. (*i.e.*, Abox axioms), are equivalent to DeLP facts of the form "$C(a)$" and "$P(a, b)$", where a and b are constants.

Class and property equivalence axioms can be replaced with a symmetrical pair of inclusion axioms, so they can be mapped to a symmetrical pair of DeLP rules as follows: the class C is equivalent to the class D (noted as $C \equiv D$) maps to the set of DeLP rules $\{C(X) \prec D(X); D(X) \prec C(X)\}$ and the property P is equivalent to the property Q (noted as $P \equiv Q$) maps to the set of DeLP rules $\{P(X, Y) \prec Q(X, Y); Q(X, Y) \prec P(X, Y)\}$. Inverse and transitivity axioms can also be translated: the property P is the inverse of the property Q (noted as $P \equiv Q^-$) maps to the set of DeLP rules $\{Q(Y, X) \prec P(X, Y); P(X, Y) \prec Q(Y, X)\}$ and the property P is transitive (noted as $P^+ \sqsubseteq P$) maps to "$P(X, Z) \prec P(X, Y) \wedge P(Y, Z)$".

4.2 Translating Class Constructors

In the previous section we showed how DL axioms correspond with DeLP rules, and how these can be used to make statements about classes and properties. In DL the classes appearing in such axioms need not be atomic, but can be complex compound expressions built up from atomic classes and properties using a variety of constructors. Next we will show how these DL expressions correspond to expressions in the body of DeLP rules. We will use C, D to denote classes, and P, Q to denote properties.

Conjunctions: A DL class can be formed by conjoining existing classes (noted as $C \sqcap D$), which corresponds to a conjunction of unary predicates. Conjunction can be directly expressed in the body of a DeLP rule. When a conjunction occurs on the left-hand side (l.h.s.) of a subclass axiom (as in $C_1 \sqcap C_2 \sqsubseteq D$) it becomes a conjunction in the body of the corresponding rule "$D(X) \prec C_1(X) \wedge C_2(X)$". When a conjunction occurs on the right-hand side (r.h.s.) of a subclass axiom (as in $C \sqsubseteq D_1 \sqcap D_2$), it becomes conjunction in the head of the rule "$D_1(X) \wedge D_2(X) \prec C(X)$"; this can be transformed (via Lloyd-Topor transformations) into a set of DeLP rules $\{(D_1(X) \prec C(X)); (D_2(X) \prec C(X))\}$.

Disjunctions: A DL class can be formed from a disjunction of existing classes (noted as $C \sqcup D$), which corresponds to a disjunction of unary predicates. When a disjunction occurs on the l.h.s. of a subclass axiom (as in $C_1 \sqcup C_2 \sqsubseteq D$), it becomes a disjunction in the body of the rule "$D(X) \prec C_1(X) \vee C_2(X)$"; this is transformed (by Lloyd-Topor) into a pair of DeLP rules $\{(D(X) \prec C_1(X)); D(X) \prec C_2(X)\}$. Notice that when a disjunction occurs on the r.h.s. of a subclass axiom, it becomes a disjunction in the head of the corresponding rule, and this cannot be represented in DeLP.

Universal restrictions: In a DL, the universal quantifier can only be used in restrictions (expressions of the form $\forall P.C$, where P must be a single primitive property but C can be a compound expression). So when a universal restriction occurs on the r.h.s. of a subclass axiom ($C \sqsubseteq \forall P.D$), it is expressed as a DeLP rule "$D(Y) \prec C(X) \wedge P(X, Y)$". When a universal restriction occurs on the l.h.s. of a subclass axiom (as in $\forall P.C \sqsubseteq D$), it is equivalent to the rule "$C(Y) \prec P(X, Y) \prec D(Y)$" which is equivalent to "$\sim P(X, Y) \vee C(Y) \prec D(Y)$", which is transformed into the set of rules $\{(D(Y) \prec \sim P(X, Y)); (D(Y) \prec C(Y))\}$.

Existential restrictions: In a DL, the existential quantifier is used in an expression of the form $\exists P.C$, where P must be a single primitive property, but C may be a compound expression. When an

existential restriction occurs on the l.h.s. of a subclass axiom (as in $\exists P.C \sqsubseteq D$), it is expressed as a conjunction in the body of a DeLP rule "$D(X) \prec P(X,Y) \wedge C(Y)$". When an existential restriction occurs on the r.h.s. of a subclass axiom, it should be expressed as a conjunction in the head of the corresponding rule, with a variable that is existentially quantified. This cannot be represented in DeLP.

4.3 A Recursive Mapping from DL to DeLP

In [11], Grosof *et al.* present a mapping from DL to a subset of Logic Programming. In this section, we will present an adaptation of such mapping for translating DL KBs into DeLP programs.

In the previous sections, we have defined all the cases regarding the translation of DL sentences and axioms into DeLP. Using those definitions as guidelines and assuming that every DL sentence is normalized w.r.t. negation, every DL axiom is mapped into one or more DeLP rules.

Definition 1 (Mapping $\mathcal{T}$ from DL to DeLP) *Let A, C, D be concepts, X, Y be variables, P, Q be relationships.*

$$
\begin{array}{lll}
\mathcal{T}(C \sqsubseteq D) & =_{df} & Th(D,X) \prec Tb(C,X) \\
Th(A,X) & =_{df} & A(X) \\
Th((C \sqcap D),X) & =_{df} & Th(C,X) \wedge Th(D,X) \\
Th((\forall R.C),X) & =_{df} & Th(C,Y) \prec R(X,Y) \\
Tb(A,X) & =_{df} & A(X) \\
Tb((C \sqcap D),X) & =_{df} & Tb(C,X) \wedge Tb(D,X) \\
Tb((C \sqcup D),X) & =_{df} & Tb(C,X) \vee Tb(D,X) \\
Tb((\exists R.C),X) & =_{df} & R(X,Y) \wedge Tb(C,Y) \\
\mathcal{T}(C \equiv D) & =_{df} & \left\{ \begin{array}{l} \mathcal{T}(C \sqsubseteq D) \\ \mathcal{T}(D \sqsubseteq C) \end{array} \right.
\end{array}
\qquad
\begin{array}{lll}
\mathcal{T}(\top \sqsubseteq \forall P.D) & =_{df} & Th(D,Y) \prec P(X,Y) \\
\mathcal{T}(\top \sqsubseteq \forall P^{-}.D) & =_{df} & Th(D,X) \prec P(X,Y) \\
\mathcal{T}(D(a)) & =_{def} & Th(D,a) \\
\mathcal{T}(P(a,b)) & =_{def} & P(a,b) \\
\mathcal{T}(P \sqsubseteq Q) & =_{df} & Q(X,Y) \prec P(X,Y) \\
\mathcal{T}(P \equiv Q) & =_{df} & \left\{ \begin{array}{l} Q(X,Y) \prec P(X,Y) \\ P(X,Y) \prec Q(X,Y) \end{array} \right. \\
\mathcal{T}(P \equiv Q^{-}) & =_{df} & \left\{ \begin{array}{l} Q(X,Y) \prec P(Y,X) \\ P(Y,X) \prec Q(X,Y) \end{array} \right. \\
\mathcal{T}(P^{+} \sqsubseteq P) & =_{df} & P(X,Z) \prec P(X,Y) \wedge P(Y,Z)
\end{array}
$$

Besides, rules of the form "$(H \wedge H') \prec B$" are rewritten as two rules "$H \prec B$" and "$H' \prec B$", rules of the form "$H \prec H' \prec B$" are rewritten as "$H \prec B \wedge H'$", and rules of the form "$H \prec (B \vee B')$" are rewritten as two rules "$H \prec B$" and "$H \prec B'$".

Example 2 *Consider the DL knowledge base $\mathcal{KB}$ shown in Example 1, the DeLP $\mathcal{P}$ in Figure 2 is obtained by applying the mapping $\mathcal{T}$ to $\mathcal{KB}$. Rules are numbered in Roman and each rule corresponds to the equivalent rule numbered in Arabic in Fig. 1 (*e.g.*, $\mathcal{T}((3)) = (iii)$). Moreover when one rule from Example 1 generates more than one rule in $\mathcal{P}$ according to $\mathcal{T}$, the resulting rules are superscripted to show their origin (*e.g.*, $\mathcal{T}((10)) = \{(x), (x'), (x")\}$).*

For the sake of example, rule (6) in Figure 1 expressing that every penguin which was operated by a genetic surgeon is a genetically altered penguin is now mapped into rule (vi) in Figure 2:

$$geneticallyAlteredPenguin(X) \prec penguin(X), isOperatedBy(X,Y), geneticSurgeon(Y)$$

which now expresses that every penguin which was operated by a genetic surgeon is usually *a genetically altered penguin.*

Moreover, rule (10) in Figure 1 saying that the set of genetic surgeons is exactly the intersection of the set of genetists and the set of surgeons is now expressed as the three rules:

$$
\begin{array}{ll}
(x) & geneticSurgeon(X) \prec genetist(X), surgeon(X); \\
(x') & genetist(X) \prec geneticSurgeon(X); \\
(x") & surgeon(X) \prec geneticSurgeon(X)
\end{array}
$$

expressing that (x) an individual is usually a genetic surgeon every time she is both a genetist and a surgeon, (x') an individual is usually a genetist if she is a genetic surgeon, and (x") an individual is usually a surgeon when she is a genetic surgeon.

$$\Delta = \left\{ \begin{array}{ll} (i) & animal(X) \prec bird(X); \\ (ii) & fly(X) \prec bird(X); \\ (iii) & bird(X) \prec eagle(X); \\ (iv) & bird(X) \prec penguin(X); \\ (v) & \sim fly(X) \prec penguin(X); \\ (vi) & geneticallyAlteredPenguin(X) \prec penguin(X), isOperatedBy(X,Y), geneticSurgeon(Y); \\ (vii) & fly(X) \prec geneticallyAlteredPenguin(X); \\ (viii) & \sim fly(X) \prec eagle(X), hasBrokenWing(X); \\ (ix) & operates(X,Y) \prec isOperatedBy(Y,X); \\ (ix') & isOperatedBy(X,Y) \prec operates(Y,X); \\ (x) & geneticSurgeon(X) \prec genetist(X), surgeon(X); \\ (x') & genetist(X) \prec geneticSurgeon(X); \\ (x") & surgeon(X) \prec geneticSurgeon(X) \end{array} \right\}$$

$$\Pi = \left\{ \begin{array}{ll} (xi) & penguin(opus); \\ (xii) & eagle(avenger); \\ (xiii) & hasBrokenWing(avenger); \\ (xiv) & genetist(frankenstein); \\ (xv) & surgeon(frankenstein); \\ (xvi) & operates(frankenstein, opus) \end{array} \right\}$$

Figure 2: DeLP knowledge base $\mathcal{P} = (\Pi, \Delta)$ obtained from $\mathcal{KB}$ via $\mathcal{T}$

5 AN APPROACH TO HANDLING INCONSISTENCIES IN ONTOLOGY DEFINITIONS BASED ON DELP

It is known that ontologies expressed in ontology languages such as OWL-DL can be expressed as equivalent DL ontologies. Although there exist implementations of DL reasoners (*e.g.*, RACER [12]), they are incapable of dealing with inconsistent ontologies. As explained above, our proposal consists of transforming an ontology expressed in a DL into an equivalent DeLP program. Thus, inconsistencies arising from inconsistent ontology definitions will be handled by the DeLP engine by performing a dialectical analysis in order to determine which conclusions arising from the derived DeLP program (and indirectly from the original ontology) are warranted. In this section, we will address the issues of reasoning about instances provided an inconsistent ontology definition, reasoning about the class structure of an inconsistent ontology definition provided no instance information is given, and finally will discuss a possible architecture for using our approach in the context of the Semantic Web.

5.1 Reasoning about Instances

As explained in Section 1, our proposal consists of transforming an ontology $O_{DL} = (Tbox, Abox)$ into an equivalent DeLP program $\mathcal{P} = (\Pi, \Delta)$ where axioms (*i.e.*, the $Tbox$) in O_{DL} will correspond to a set Δ of defeasible rules in $\mathcal{P}$ while information about individuals in O_{DL} (*i.e.*, the $Abox$) will correspond to the set Π of facts in $\mathcal{P}$.

As the set Π comes from the transformation of the $Abox$, it is solely composed of facts. For the sake of simplicity we will asume that Π is consistent[3] as required for the DeLP framework. Clearly, in case Π is inconsistent, the pair of conflicting literals can be easily detected and removed.

When the original DL ontology O_{DL} is consistent, the resulting DeLP program $\mathcal{P}$ is also consistent. In the case that O_{DL} is inconsistent, this situation is reflected by the fact that contradicting facts can be entailed by O_{DL}. When considering the translated ontology into $\mathcal{P} = (\Pi, \Delta)$, the situation

[3]That is, there is neither a pair of facts $C(a)$ and $\sim C(a)$ nor $R(a,b)$ and $\sim R(a,b)$ where C is a concept name, R is a property name, and a and b are individual constants.

of O_{DL} being inconsistent is reflected into an inconsistent $\Pi \cup \Delta$. This situation will lead to the derivation of conflicting literals as shown in the next example.

Example 3 *Consider the DL ontology presented in Example 1, this ontology is inconsistent as* $\mathcal{KB} \models \{fly(opus), \neg fly(opus)\}$ *and* $\mathcal{KB} \models \{fly(avenger), \neg fly(avenger)\}$. *Considering now the DeLP program* $\mathcal{P}$ *presented in Example 2 and obtained from* $\mathcal{KB}$, *this situation is reflected by the existence of arguments* $\langle \mathcal{A}_1, fly(avenger)\rangle$ *and* $\langle \mathcal{A}_2, \sim fly(avenger)\rangle$ *as well as* $\langle \mathcal{B}_1, fly(opus)\rangle$ *and* $\langle \mathcal{B}_2, \sim fly(opus)\rangle$.[4]

As explained in Section 3, in DeLP given a query Q, consideration of conflicting arguments on behalf and against Q leads to a process known as dialectical argumentation in which defeaters for arguments favoring Q have to be taken into account as well as defeaters for these defeaters and so on. In the case of having an inconsistent DL ontology O_{DL} and given a query Q, the resulting equivalent DeLP program will be fed into the DeLP engine to carry out such dialectical analysis. The following example depicts the above described situation.

Example 4 *Inferences in the DL KB* $\mathcal{KB}$, *such as* $\mathcal{KB} \models fly(opus)$, *are modeled in DeLP program* $\mathcal{P}$ *as* $\mathcal{P} \vdash fly(opus)$. *However, as the derived program* $\mathcal{P}$ *is inconsistent, we have that both* $\mathcal{P} \vdash fly(opus)$ *and* $\mathcal{P} \vdash \sim fly(opus)$, *leading to the construction of conflicting arguments.*

Next we will show the arguments arising from $\mathcal{P}$ *and will characterize their interactions in the dialectical analysis that arises when considering them. There exists an argument* $\mathcal{A}_1$ *supporting the defeasible conclusion that Avenger flies,* i.e., $\langle \mathcal{A}_1, fly(avenger)\rangle$ *where:*

$$\mathcal{A}_1 \;=\; \{(fly(avenger) \prec bird(avenger)); (bird(avenger) \prec eagle(avenger))\}$$

Assuming that the rule comparison establishes that $(8) \succ (2)$, *this argument is defeated by an argument* $\langle \mathcal{A}_2, \sim fly(avenger)\rangle$ *supporting that Avenger does not fly because he has a broken wing, where:*

$$\mathcal{A}_2 \;=\; \{\sim fly(avenger) \prec eagle(avenger), hasBrokenWing(avenger)\}$$

Argument $\mathcal{A}_2$ *has no defeaters, argument* $\mathcal{A}_1$ *is therefore defeated and it is marked as a* D*-node. The corresponding dialectical tree is depicted in Figure 3.(i).*

Likewise there exists an argument $\mathcal{B}_1$ *supporting the defeasible conclusion that Opus flies,* i.e., $\langle \mathcal{B}_1, fly(opus)\rangle$ *where:*

$$\mathcal{B}_1 \;=\; \{(fly(opus) \prec bird(opus)); (bird(opus) \prec penguin(opus))\}$$

Another argument $\langle \mathcal{B}_2, \sim fly(opus)\rangle$ *can be derived from* $\mathcal{P}$, *supporting the conclusion that Opus does not fly, with:*

$$\mathcal{B}_2 \;=\; \{\sim fly(opus) \prec penguin(opus)\}$$

Argument $\mathcal{B}_2$ *defeats* $\mathcal{B}_1$ *provided that the rule comparison criterion establishes that* $(5) \succ (2)$. *However, provided that* $(7) \succ (5)$, *argument* $\mathcal{B}_2$ *is defeated by another argument* $\langle \mathcal{B}_3, fly(opus)\rangle$ *which reinstates argument* $\mathcal{B}_1$, *where:*

$$\begin{aligned}\mathcal{B}_3 \;=\; & \{(fly(opus) \prec geneticallyAlteredPenguin(opus));\\ & (geneticallyAlteredPenguin(opus) \prec penguin(opus), isOperatedBy(opus, frankenstein),\\ & \qquad\qquad geneticSurgeon(frankenstein));\\ & (isOperatedBy(opus, frankenstein) \prec operates(frankenstein, opus));\\ & (geneticSurgeon(frankenstein) \prec genetist(frankenstein), surgeon(frankenstein))\}\end{aligned}$$

[4]Notice that this also happens because of the existence of arguments $\langle \mathcal{B}_2, \sim fly(opus)\rangle$ and $\langle \mathcal{B}_3, fly(opus)\rangle$.

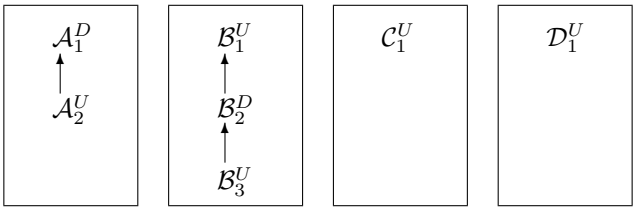

Figure 3: Dialectical trees for: (i) $fly(avenger)$, (ii) $fly(opus)$, (iii) $animal(avenger)$, and (iv) $animal(opus)$

Hence the associated dialectical tree for $fly(opus)$ has three nodes, with the root labelled as U-node (see Figure 3.(ii)). The original argument for $fly(opus)$ is therefore warranted.

There exist two more arguments $\langle \mathcal{C}_1, animal(avenger) \rangle$ and $\langle \mathcal{D}_1, animal(opus) \rangle$ supporting the defeasible conclusions that both Avenger and Opus are animals resp., where:

$$\begin{aligned} \mathcal{C}_1 &= \{(animal(avenger) \prec bird(avenger)); (bird(avenger) \prec eagle(avenger))\} \\ \mathcal{D}_1 &= \{(animal(opus) \prec bird(opus)); (bird(penguin) \prec penguin(opus))\} \end{aligned}$$

These two arguments have no defeaters, they are therefore warranted and the resulting dialectical trees will have a unique node, as depicted in Figure 3.(iii–iv).

5.2 Reasoning about Class Structure

Given an ontology $O_{DL} = (Tbox, Abox)$ with $Abox = \emptyset$, the resulting DeLP program $\mathcal{P} = (\Pi, \Delta)$ obtained when applying the transformation function $\mathcal{T}$ presented above has no facts (*i.e.*, $\Pi = \emptyset$). Thus, the DeLP engine will not be able to infer any information from Δ alone. In the case that $Abox = \emptyset$ and the ontology still was unsatisfiable, it would be desirable to be able to detect this situation in the framework of logic programming.

The solution to this problem consists of translating the $Tbox$ to an equivalent AnsProlog$^{\neg,\bot}$ program [2] enriched with information regarding class unsatisfiability. Each DL axiom in $Tbox$ is transformed according to Grosof *et al.*'s mapping [11]. Besides for each class name, a constraint is added to prevent the existence of complementary literals.

Example 5 *Consider again the DL knowledge base $\mathcal{KB}$ presented in Example 1. Transforming the $Abox$ into AnsProlog$^{\neg,\bot}$ results in the program $\mathcal{P}_{LP}$ presented in Figure 4. Every rule equivalent to the respective rule enumerated with lowercase Romans in Figure 1 is enumerated in uppercase Romans.*

5.3 An Architecture for Handling Inconsistent Ontologies in the Web

In this section we will present a possible architecture which integrates a web browser with a web form for interaction with a human user. The web browser will ultimately access a remote database stored in the web whose data is defined according to some ontology definitions that might be inconsistent. Our proposal for inconsistent ontology management therefore consists in proposing a web service to which a web browser can connect on behalf of a human user who wants to query a certain database located somewhere in the web. The architecture for the approach is depicted in Figure 5.

(I) $animal(X) \leftarrow bird(X);$
(II) $fly(X) \leftarrow bird(X);$
(III) $bird(X) \leftarrow eagle(X);$
(IV) $bird(X) \leftarrow penguin(X);$
(V) $\neg fly(X) \leftarrow penguin(X);$
(VI) $geneticallyAlteredPenguin(X) \leftarrow penguin(X), isOperatedBy(X, Y), geneticSurgeon(Y);$
$\ldots$
(X) $geneticSurgeon(X) \leftarrow genetist(X), surgeon(X);$
(X') $genetist(X) \leftarrow geneticSurgeon(X);$
$(X")$ $surgeon(X) \leftarrow geneticSurgeon(X);$
$\bot \leftarrow animal(X), \neg animal(X)$
$\bot \leftarrow bird(X), \neg bird(X)$
$\bot \leftarrow fly(X), \neg fly(X)$
$\bot \leftarrow penguin(X), \neg penguin(X)$
$\ldots$

Figure 4: AnsProlog$^{\neg,\bot}$ program $\mathcal{P}_{LP}$ obtained from $\mathcal{KB}$ via Grosof *et al.*'s mapping

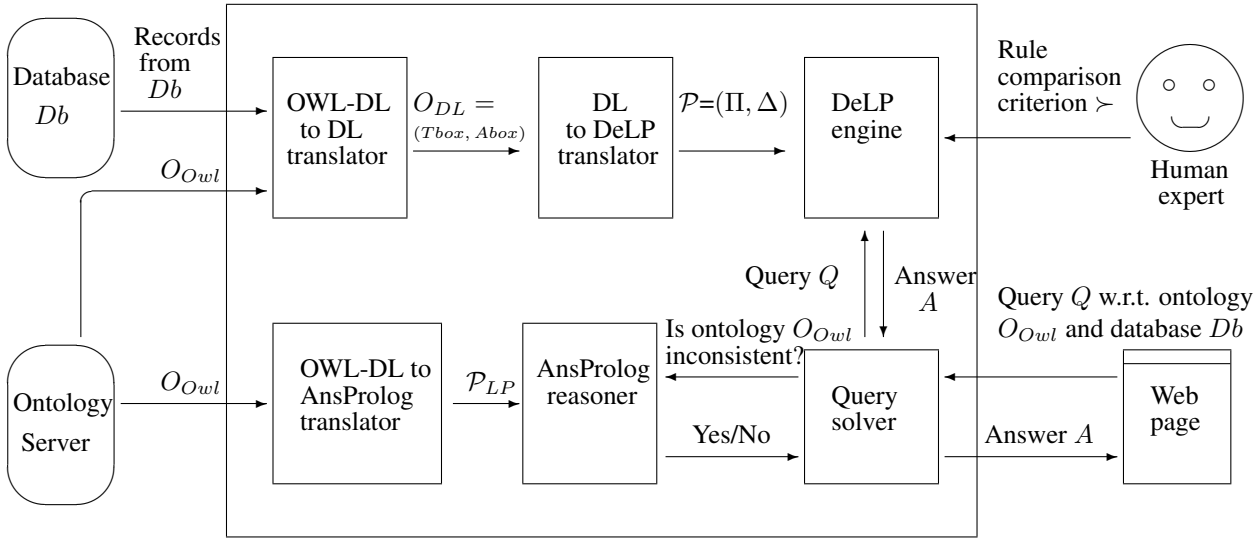

Figure 5: A possible architecture for handling inconsistent ontology definitions in DeLP

We will assume that the web browser is able to query a remote database Db in the web where the information contained in its data records must be interpreted according to an ontology definition O_{Owl} also accesible in the web. Incidentally we will also assume the ontology definition O_{Owl} will be stored in an ontology server.

The proposed system works as follows. The user issues through the web browser a query Q w.r.t. to database Db and ontology O_{Owl}. This query is processed by the query solver who asks the AnsProlog reasoner if O_{Owl} is consistent which is retrieved by the OWL-DL to AnsProlog translator from the ontology server. If the ontology O_{Owl} is inconsistent, it is translated to a DeLP program $\mathcal{P} = (\Pi, \Delta)$ along with the records from database Db by the OWL-DL to DL and DL to DeLP translators. Using $\mathcal{P}$, the DeLP engine performs a dialectical analysis to determine the epistemic status A of Q w.r.t. to $\mathcal{P}$. Finally the epistemic status of Q is returned as answer A which will be displayed in the web browser.

In order to perform the dialectical process of Q w.r.t. $\mathcal{P}$, we will also assume that there exists a human expert who provides the comparison criterion $\succ$ for performing rule comparison into the DeLP engine.

6 RELATED WORK

RACER [12] implements a TBox and ABox reasoner for the logic $\mathcal{SHIQ}$, it was also the first full-fledged ABox description logic system for a very expressive logic and is based on optimized sound and complete algorithms. In constrast to our proposal, it is not able to deal with inconsistent ontology definitions.

Grosof *et al.* [11] show how to interoperate, semantically and inferentially, between the leading Semantic Web approaches to rules (RuleML Logic Programs) and ontologies (OWL/DAML+OIL Description Logic) via analyzing their expressive intersection. To do so, we define a new intermediate knowledge representation (KR) contained within this intersection: Description Logic Programs (DLP), and the closely related Description Horn Logic (DHL) which is an expressive fragment of first-order logic (FOL). They show how to perform DLP-fusion: the bidirectional translation of premises and inferences (including typical kinds of queries) from the DLP fragment of DL to LP, and vice versa from the DLP fragment of LP to DL. Part of our article is based on Grosof *et al.* work as we found the algorithm for translating DL ontologies into DeLP on their work. However, as Grosof *et al.*work uses standard Prolog rules, they are not able to deal with inconsistent DL knowledge bases as our proposal does.

In [13], Heymans and Vermier extend the description logic $\mathcal{SHOQ}(D)$ with a preference order on the axioms. With this strict partial order certain axioms can be overruled, if defeated with more preferred ones. They also impose a preferred model semantics, thus effectively introducing nonmonotonicity into $\mathcal{SHOQ}(D)$. They argue that since a description logic can be viewed as an ontology language, or a proper translation of one, they obtain a defeasible ontology language. Similarly to Heymans and Vermier's work we allow for inferencing from inconsistent ontologies by considering subsets of of the original KB (arguments), comparing them in terms of a rule comparison criterion. However, we choose to translate to the original DL KB into the DeLP language.

In [7], Eiter *et al.* propose a combination of logic programming under the answer set semantics with the description logics $\mathcal{SHIF}(D)$ and $\mathcal{SHOIN}(D)$, which underlie the Web ontology languages OWL Lite and OWL DL, resp. This combination allows for building rules on top of ontologies but also, to a limited extent, building ontologies on top of rules. In contrast to our approach, they keep separated rules and ontologies and handle exceptions by codifying them explicitly in programs under answer set semantics.

7 CONCLUSIONS

We have presented a novel argument-based approach for handling inconsistent ontology definitions in the Semantic Web. As discussed in the introduction, given an ontology expressed in a DL we proposed translating it into an equivalent DeLP program. Then given a query posed w.r.t. the DL ontology, it will be answered w.r.t. to the DeLP program.

Our approach is not able to deal with axioms that requires the construction of rules with a disjunction in the head which are not currently supported by DeLP, a possible extension to this work would consist in enhancing DeLP with the possibility of handling disjunctions in the head of the rules.

Other issue associated with the approach that needs to be addresed is given by the mapping of equivalence DL axioms into DeLP rules. Currently a DL axiom of the form '$C \equiv D$' generates two rules of the form '$C(X) \prec D(X)$' and '$D(X) \prec C(X)$'. This situation could clearly produce loops during the argument construction when solving queries in actual DeLP programs.

One of the goals of the Semantic Web initiative consists of providing methods of entailing information from ontologies that scale to the size of the web. Cecchi *et al.* [4] have proved that the

complexity of the decision problems of determining whether a set of defeasible rules is an argument for a literal under a DeLP program and determining whether there exists an argument for a given literal are P-complete and NP, respectively. Other issue to address involves eliminating the assistance of a human user in order to provide a rule comparison criterion for performing the dialectical process. Part of our current research is focused on these issues.

ACKNOWLEDGMENTS

The authors would like to thank Pablo R. Fillottrani for helpful discussions during the early stages of this paper. This research was funded by Agencia Nacional de Promoción Científica y Tecnológica (PICT 2002 No. 13.096, PICT 2003 No. 15.043, PAV 2004 076), by CONICET (Argentina), by projects TIC2003-00950 and TIN2004-07933-C03-03 (MCyT, Spain) and by Ramón y Cajal Program (MCyT, Spain). The authors would also like to thank the anonymous reviewers for helpful comments to improve the final version of this paper.

REFERENCES

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

[2] Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

[3] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scient. American*, 2001.

[4] Laura A. Cecchi, Pablo R. Fillottrani, and Guillermo R. Simari. On Complexity of DeLP through Game Semantics. In J. Dix and A. Hunter, editors, *11th. Intl. Workshop on Nonmonotonic Reasoning*, pages 386–394, 2006.

[5] C. Chesñevar and A. Maguitman. ARGUENET: An Argument-Based Recommender System for Solving Web Search Queries. In *Proc. of the 2nd IEEE Intl. IS-2004 Conference. Varna, Bulgaria*, pages 282–287, June 2004.

[6] C. Chesñevar, A. Maguitman, and R. Loui. Logical Models of Argument. *ACM Computing Surveys*, 32(4):337–383, December 2000.

[7] Thomas Eiter, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Combining Answer Set Programming with Description Logics for the Semantic Web. *KR 2004*, pages 141–151, 2004.

[8] A. García and G. Simari. Defeasible Logic Programming an Argumentative Approach. *Theory and Prac. of Logic Program.*, 4(1):95–138, 2004.

[9] S. Gómez and C. Chesñevar. A Hybrid Approach to Pattern Classification Using Neural Networks and Defeasible Argumentation. In *Proc. of 17th Intl. FLAIRS Conference. Miami, Florida, USA*, pages 393–398. American Assoc. for Art. Intel., May 2004.

[10] Sergio Alejandro Gómez, Carlos Iván Chesñevar, and Guillermo Ricardo Simari. Incorporating Defeasible Knowledge and Argumentative Reasoning in Web-based Forms. In *In 19th Intl. Joint Conf. in Artificial Intelligence (IIJCAI 2005). Edimburgh, UK, July 2005*, pages 9–16, 2005.

[11] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description Logic Programs: Combining Logic Programs with Description Logics. *WWW2003, May 20-24, Budapest, Hungary*, 2003.

[12] Volker Haarslev and Ralf Möller. RACER System Description. Technical report, University of Hamburg, Computer Science Department, 2001.

[13] S. Heymans and D. Vermeir. A Defeasible Ontology Language. *CoopIS/DOA/ODBASE 2002*, pages 1033–1046, 2002.

[14] Deborah L. McGuiness and Frank van Harmelen. OWL Web Ontology Language Overview, 2004. http://www.w3.org/TR/owl-features/.

[15] Henry Prakken and Gerard Vreeswijk. Logical Systems for Defeasible Argumentation. In D. Gabbay and F.Guenther, editors, *Handbook of Philosophical Logic*, pages 219–318. Kluwer Academic Publishers, 2002.

[16] G. Simari and R. Loui. A Mathematical Treatment of Defeasible Reasoning and its Implementation. *Artificial Intelligence*, 53:125–157, 1992.