An Argumentative Approach to Reasoning with Inconsistent Ontologies

Sergio Alejandro Gómez[1] **Carlos Iván Chesñevar**[1,2]
Guillermo Ricardo Simari[1]

[1] Artificial Intelligence Research and Development Laboratory,
Department of Computer Science and Engineering,
Universidad Nacional del Sur,
Av. Alem 1253 - (8000) Bahía Blanca, Argentina,
Email: {sag,cic,grs}@cs.uns.edu.ar
[2]CONICET (National Council of Scientific and Technical Research), Argentina

Abstract

Standard approaches to reasoning with Description Logics (DL) ontologies require them to be consistent. However, as ontologies are complex entities and sometimes built upon other imported ontologies, inconsistencies can arise. In this paper, we present a framework for reasoning with inconsistent DL ontologies. Our proposal involves expressing DL ontologies as Defeasible Logic Programs (DeLP). Given a query posed w.r.t. an inconsistent ontology, a dialectical analysis will be performed on a DeLP program obtained from such ontology where all arguments in favor and against the final answer of the query will be taken into account. We also present an application to ontology integration based on the global-as-view approach.

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

1 Introduction and Motivations

The *Semantic Web* (Berners-Lee et al. 2001) 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* (Gruber 1993).

As proposed by the World Wide Web Consortium[1], ontology definitions are meant to be written in an *ontology description language* such as OWL (McGuiness & van Harmelen 2004), whose subset known as OWL-DL is based on Description Logics (DL) (Baader et al. 2003). Although ontology definitions expressed in DL can be processed with existing DL reasoners (*e.g.* RACER (Haarslev & Möller 2001)), such DL reasoners are incapable of dealing with *inconsistent* ontology definitions.

This situation is particularly important in the Semantic Web setting, where ontologies are complex entities prone to suffer inconsistencies (Huang et al. 2005). A particular source of inconsistency is related to the use of imported ontologies when the knowledge engineer has no authority to correct them. As these imported ontologies are usually developed independently, their combination could also result in inconsistencies. The problem of combining two or more different ontologies in order to obtain a unified, consistent ontology is known as *ontology integration* (Klein 2001). One kind of such integration is known as *global-as-view* integration (Calvanese et al. 2001), where a global ontology is used as a view of local ontologies that define the actual data.

There are two main ways to deal with inconsistency in ontologies (Huang et al. 2005): one is to diagnose and repair it when it is encountered; another is to avoid the inconsistency and to apply a non-standard inference relation to obtain meaningful answers. Although there are existing approaches for the former (*e.g.* identifying the minimally unsatisfiable sub-ontologies or calculating the maximally satisfiable sub-ontologies), in this work we propose to use *defeasible argumentation* (Chesñevar et al. 2000, Prakken & Vreeswijk 2002) to focus on the latter. Grosof et al. (2003) have 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 (KB) codified as a set of Horn-like clauses called DeLP programs (García & Simari 2004). When presented with a query, DeLP performs a *dialectical* process in which all *arguments* in favor and against a conclusion are considered; arguments regarded as ultimately *undefeated* will be considered *warranted.*

In this article we propose a framework for representing possibly inconsistent DL ontologies. Reasoning in such ontologies will be carried out by means of a dialectical analysis. Our proposal involves mapping DL ontologies into a DeLP program. That is, given a DL ontology Σ_{DL}, provided Σ_{DL} satisfies certain restrictions, it will be translated into a DeLP program Σ_{DeLP}. Given a query ϕ, a dialectical process will be performed to determine if ϕ is warranted w.r.t. Σ_{DeLP}. Besides, we apply our proposal to perform global-as-view integration when the involved ontologies can be potentially inconsistent.

The rest of the article is structured as follows. Section 2 introduces the fundamentals of Description Logics. Section 3 briefly explains the Defeasible Logic Programming formalism. Section 4 explains how DL ontologies are going to be translated into DeLP. Section 5 introduces δ-ontologies, a particular kind of DL ontologies amenable for its treatment within DeLP. Section 6 presents an application to on-

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 Project TIN2006-15662-C02-01 (MEC, Spain), Project 24/ZN10 (SGCyT, UNS).

[1]www.w3c.org

tology integration based on defeasible argumentation. Section 7 enumerates some properties of the proposed approach. Section 8 discusses related work. Finally Section 9 concludes.

2 Description Logics

Description Logics (DL) are a well-known family of knowledge representation formalisms (Baader et al. 2003). 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 transitive roles noted as P^- and P^+, resp.

A DL *ontology* $\Sigma = (T, A)$ consists of two finite and mutually disjoint sets: the Tbox T which introduces the *terminology* and the Abox A 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 $a : C$ and *role assertions* of the type $\langle a, b \rangle : R$, where C is a concept description, R is a role name, and a and b are individual names. An interpretation I *satisfies* the assertion $a : C$ iff $a^I \in C^I$, and it *satisfies* $\langle a, b \rangle : R$ 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 ontology Σ iff it satisfies every statement in Σ. A DL ontology Σ *entails* a DL statement ϕ, written as $\Sigma \models \phi$, iff every model of Σ is a model of ϕ.

3 Defeasible Logic Programming

Defeasible Logic Programming (DeLP) (García & Simari 2004) provides a language for knowledge representation and reasoning that uses *defeasible argumentation* (Chesñevar et al. 2000, Prakken & Vreeswijk 2002, Simari & Loui 1992) to decide between contradictory conclusions through a *dialectical analysis.* Recent research has shown that DeLP provides a suitable framework for building real-world applications 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.

Definition 1 (Strict, defeasible and DeLP programs) $\mathcal{L}_{DeLP} =_{df} \mathcal{L}_{DeLP_\Pi} \cup \mathcal{L}_{DeLP_\Delta}$ *is the language of DeLP programs, where* $\mathcal{L}_{DeLP_\Pi}$ *is the language of DeLP programs formed by strict rules* $B \leftarrow A_1, \ldots, A_n$ *with* $(n \geq 1)$ *and facts* B *(i.e., rules where* $n = 0$*), and* $\mathcal{L}_{DeLP_\Delta}$ *is the language of DeLP programs formed only by defeasible rules* $B \prec A_1, \ldots, A_n$ *with* $(n \geq 1)$.

Literals can be positive or negative. The complement of a literal L (noted as $\overline{L}$) is p if $L = \sim p$ and $\sim p$ if $L = p$. Deriving literals in DeLP results in the construction of *arguments.* An argument $\mathcal{A}$ is a (possibly empty) set of ground defeasible rules that together with the set Π provides a logical proof for a given literal Q, satisfying the additional requirements of *non-contradiction* and *minimality.* Formally:

Definition 2 (Argument) *Given a DeLP program* $\mathcal{P}$*, an* argument $\mathcal{A}$ *for a query* Q*, denoted* $\langle \mathcal{A}, Q \rangle$*, is a subset of ground instances of defeasible rules in* $\mathcal{P}$*, such that: (1) there exists a* defeasible derivation *for* Q *from* $\Pi \cup \mathcal{A}$*; (2)* $\Pi \cup \mathcal{A}$ *is non-contradictory (i.e.,* $\Pi \cup \mathcal{A}$ *does not entail two complementary literals* P *and* $\sim P$*), and, (3) there is no* $\mathcal{A}' \subseteq \mathcal{A}$ *such that there exists a defeasible derivation for* Q *from* $\Pi \cup \mathcal{A}'$*. An argument* $\langle \mathcal{A}_1, Q_1 \rangle$ *is a* sub-argument *of another argument* $\langle \mathcal{A}_2, Q_2 \rangle$ *if* $\mathcal{A}_1 \subseteq \mathcal{A}_2$*.*

The notion of defeasible derivation corresponds to the usual query-driven SLD derivation used in logic programming, performed by backward chaining on both strict and defeasible rules; in this context a negated literal $\sim P$ is treated just as a new predicate name *no_P*. Minimality imposes a kind of 'Occam's razor principle' (Simari & Loui 1992) on argument construction. The non-contradiction requirement forbids the use of (ground instances of) defeasible rules in an argument $\mathcal{A}$ whenever $\Pi \cup \mathcal{A}$ entails two complementary literals. The notion of contradiction is captured by the notion of counterargument.

Definition 3 (Counterargument. Defeat) *An argument* $\langle \mathcal{A}_1, Q_1 \rangle$ *is a* counterargument *for an argument* $\langle \mathcal{A}_2, Q_2 \rangle$ *iff there is a subargument* $\langle \mathcal{A}, Q \rangle$ *of* $\langle \mathcal{A}_2, Q_2 \rangle$ *such that the set* $\Pi \cup \{Q_1, Q\}$ *is contradictory. An argument* $\langle \mathcal{A}_1, Q_1 \rangle$ *is a* defeater *for an argument* $\langle \mathcal{A}_2, Q_2 \rangle$ *if* $\langle \mathcal{A}_1, Q_1 \rangle$ *counterargues* $\langle \mathcal{A}_2, Q_2 \rangle$*, and* $\langle \mathcal{A}_1, Q_1 \rangle$ *is preferred over* $\langle \mathcal{A}_2, Q_2 \rangle$ *w.r.t. a* preference criterion $\preceq$ *on conflicting arguments. Such criterion is defined as a partial order* $\preceq \subseteq Args(\mathcal{P}) \times Args(\mathcal{P})$*. The argument* $\langle \mathcal{A}_1, Q_1 \rangle$ *will be called a* proper defeater *for* $\langle \mathcal{A}_2, Q_2 \rangle$ *iff* $\langle \mathcal{A}_1, Q_1 \rangle$ *is strictly preferred over* $\langle \mathcal{A}, Q \rangle$ *w.r.t.* $\preceq$*; if* $\langle \mathcal{A}_1, Q_1 \rangle$ *and* $\langle \mathcal{A}, Q \rangle$ *are unrelated to each other will be called a* blocking defeater *for* $\langle \mathcal{A}_2, Q_2 \rangle$*.*

Generalized specificity (Simari & Loui 1992) is typically used as a syntax-based criterion among conflicting arguments. However, other alternative partial orders could also be valid.

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. An *argumentation line* starting in an argument $\langle \mathcal{A}_0, Q_0 \rangle$ is a sequence $[\langle \mathcal{A}_0, Q_0 \rangle, \langle \mathcal{A}_1, Q_1 \rangle,$

$\langle \mathcal{A}_2, Q_2 \rangle$, ..., $\langle \mathcal{A}_n, Q_n \rangle$ ...] that can be thought of as an exchange of arguments between two parties, a *proponent* (evenly-indexed arguments) and an *opponent* (oddly-indexed arguments). Each $\langle \mathcal{A}_i, Q_i \rangle$ is a defeater for the previous argument $\langle \mathcal{A}_{i-1}, Q_{i-1} \rangle$ in the sequence, $i > 0$. In order to avoid *fallacious* reasoning, dialectics imposes additional constraints on such an argument exchange to be considered rationally *acceptable*. Given a DeLP program $\mathcal{P}$ and an initial argument $\langle \mathcal{A}_0, Q_0 \rangle$, the set of all acceptable argumentation lines starting in $\langle \mathcal{A}_0, Q_0 \rangle$ accounts for a whole dialectical analysis for $\langle \mathcal{A}_0, Q_0 \rangle$ (*i.e.*, all possible dialogues about $\langle \mathcal{A}_0, Q_0 \rangle$ between proponent and opponent), formalized as a *dialectical tree*.

Nodes in a dialectical tree $\mathcal{T}_{\langle \mathcal{A}_0, Q_0 \rangle}$ can be marked as *undefeated* and *defeated* nodes (U-nodes and D-nodes, resp.). A dialectical tree will be marked as an AND-OR tree: all leaves in $\mathcal{T}_{\langle \mathcal{A}_0, Q_0 \rangle}$ will be marked U-nodes (as they have no defeaters), and every inner node is to be marked as *D-node* iff it has at least one U-node as a child, and as *U-node* otherwise. An argument $\langle \mathcal{A}_0, Q_0 \rangle$ is ultimately accepted as valid (or *warranted*) w.r.t. a DeLP program $\mathcal{P}$ iff the root of its associated dialectical tree $\mathcal{T}_{\langle \mathcal{A}_0, Q_0 \rangle}$ is labeled as *U-node*.

Given a DeLP program $\mathcal{P}$, solving a query Q w.r.t. $\mathcal{P}$ accounts for determining whether Q is supported by (at least) one warranted argument. Different doxastic attitudes can be distinguished as follows: *Yes*, accounts for believing Q iff there is at least one warranted argument supporting Q on the basis of $\mathcal{P}$; *No*, accounts for believing $\sim Q$ iff there is at least one warranted argument supporting $\sim Q$ on the basis of $\mathcal{P}$; *Undecided*, neither Q nor $\sim Q$ are warranted w.r.t. $\mathcal{P}$, and *Unknown*, Q does not belong to the signature of $\mathcal{P}$.

4 Expressing DL Ontologies in DeLP

In the presence of inconsistent ontologies, traditional DL reasoners (such as RACER (Haarslev & Möller 2001)) issue an error message and stop further processing. Thus the burden of repairing the ontology (*i.e.*, making it consistent) is on the knowledge engineer. We are interested in coping with inconsistencies such that the task of dealing with them is automatically solved by the reasoning system. We propose using DeLP to perform such a task.

Grosof et al. (2003) show that the processing of ontologies can be improved by the use of techniques from the area of logic programming; they have identified a subset of DL languages that can be effectively mapped into a Horn-clause logics. Our proposal is in part based on such research by adapting it to the DeLP framework. We therefore propose translating a DL ontology $\Sigma = (T_S \cup T_D, A)$, with $T_S \cap T_D = \emptyset$, into a DeLP program $\mathcal{P} = (\Pi, \Delta)$ by means of a mapping $\mathcal{T}$ such that $\mathcal{P} = \mathcal{T}(\Sigma)$, where $\Pi = \mathcal{T}_\Pi(T_S) \cup \mathcal{T}_\Pi(A)$ and $\Delta = \mathcal{T}_\Delta(T_D)$. We will consider the Tbox as partitioned into two disjoint sets—a *strict terminology* T_S and a *defeasible terminology* T_D. Intuitively the set Π of strict rules in $\mathcal{P}$ will correspond to the Abox A joined with T_S in Σ, and the set Δ of defeasible rules will correspond to T_D in Σ.

In the rest of this section, we will explain how to achieve the translation of DL ontologies into DeLP programs. For clarity, strict rules of the form "$H \leftarrow B_1, \ldots, B_n$" will be sometimes written as "$H \leftarrow B_1 \wedge \ldots \wedge B_n$" and defeasible rules "$H \prec B_1, \ldots, B_n$" as "$H \prec B_1 \wedge \ldots \wedge B_n$". As noted by Grosof et al. (2003), for DL sentences to be mapped into Horn-logic rules, they must satisfy certain constraints. Conjunction and universal restrictions appearing in the right-hand side of inclusion axioms can be mapped to heads of rules (called $\mathcal{L}_h$-classes). In contrast, conjunction, disjunction and existential restriction can be mapped to rule bodies whenever they occur in the left-hand side of inclusion axioms (called $\mathcal{L}_b$-classes). As equality axioms "$C \equiv D$" are interpreted as two inclusion axioms "$C \sqsubseteq D$" and "$D \sqsubseteq C$", they must belong to the intersection of $\mathcal{L}_h$ and $\mathcal{L}_b$.

Definition 4 ($\mathcal{L}_h$ language. $\mathcal{L}_h$-classes. $\mathcal{L}_b$ language. $\mathcal{L}_b$-classes. $\mathcal{L}_{hb}$ language. $\mathcal{L}_{hb}$-classes (adapted from (Grosof et al. 2003))) *Let A be an atomic class name, C and D class expressions, and R a property. In the $\mathcal{L}_h$ language, $C \sqcap D$ is a class, and $\forall R.C$ is also a class. Class expressions in $\mathcal{L}_h$ are called* $\mathcal{L}_h$-classes. *In the $\mathcal{L}_b$ language, $C \sqcup D$ is a class, and $\exists R.C$ is a class too. Class expressions in $\mathcal{L}_b$ are called* $\mathcal{L}_b$-classes. *The $\mathcal{L}_{hb}$ language is defined as the intersection of $\mathcal{L}_h$ and $\mathcal{L}_b$. Class expressions in $\mathcal{L}_{hb}$ are called* $\mathcal{L}_{hb}$-classes.

We now define the mapping from DL to DeLP. Without losing generality, we assume that ontology definitions are normalized w.r.t. negation. That is, negations in class expressions are *shifted* inwards using De Morgan's rules and well-known relations between existential and value restrictions (Baader et al. 2003).

First we show how to map DL axioms into defeasible rules. As mentioned in Section 3, defeasible rules are meant to represent possibly inconsistent information. Thus DL axioms in defeasible terminologies are going to be interpreted as *default* class inclusions.

Definition 5 ($\mathcal{T}_\Delta$ mapping from DL sentences to DeLP defeasible rules) *Let A, C, D be concepts, X, Y variables, P, Q properties. The $\mathcal{T}_\Delta : 2^{\mathcal{L}_{DL}} \rightarrow 2^{\mathcal{L}_{DeLP_\Delta}}$ mapping is defined in Fig. 1. Besides, rules of the form "$(H_1 \wedge H') \prec B$" are rewritten as two rules "$H_1 \prec B$" and "$H_2 \prec B$", rules of the form "$H_1 \prec H_2 \prec B$" are rewritten as "$H_1 \prec B \wedge H_2$", and rules of the form "$H \prec (B_1 \vee B_2)$" are rewritten as two rules "$H \prec B_1$" and "$H_1 \prec B_2$".*

$$
\begin{array}{rcl}
\mathcal{T}_\Delta(\{C \sqsubseteq D\}) & =_{df} & \left\{ \; T_h(D,X) \prec T_b(C,X) \; \right\}, \\
 & & \text{if } C \text{ is an } \mathcal{L}_b\text{-class and } D \text{ an } \mathcal{L}_h\text{-class} \\
\mathcal{T}_\Delta(\{C \equiv D\}) & =_{df} & \mathcal{T}_\Delta(\{C \sqsubseteq D\}) \cup \mathcal{T}_\Delta(\{D \sqsubseteq C\}), \\
 & & \text{if } C \text{ and } D \text{ are } \mathcal{L}_{hb}\text{-classes} \\
\mathcal{T}_\Delta(\{\top \sqsubseteq \forall P.D\}) & =_{df} & \left\{ \; T_h(D,Y) \prec P(X,Y) \; \right\}, \\
 & & \text{if } D \text{ is an } \mathcal{L}_h\text{-class} \\
\mathcal{T}_\Delta(\{\top \sqsubseteq \forall P^-.D\}) & =_{df} & \left\{ \; T_h(D,X) \prec P(X,Y) \; \right\}, \\
 & & \text{if } D \text{ is an } \mathcal{L}_h\text{-class} \\
\mathcal{T}_\Delta(\{P \sqsubseteq Q\}) & =_{df} & \left\{ \; Q(X,Y) \prec P(X,Y) \; \right\} \\
\mathcal{T}_\Delta(\{P \equiv Q\}) & =_{df} & \left\{ \begin{array}{l} Q(X,Y) \prec P(X,Y) \\ P(X,Y) \prec Q(X,Y) \end{array} \right\} \\
\mathcal{T}_\Delta(\{P \equiv Q^-\}) & =_{df} & \left\{ \begin{array}{l} Q(X,Y) \prec P(Y,X) \\ P(Y,X) \prec Q(X,Y) \end{array} \right\} \\
\mathcal{T}_\Delta(\{P^+ \sqsubseteq P\}) & =_{df} & \left\{ \; P(X,Z) \prec P(X,Y) \wedge P(Y,Z) \; \right\} \\
\mathcal{T}_\Delta(\{s_1, \ldots, s_n\}) & =_{df} & \bigcup_{i=1}^{n} \{\mathcal{T}_\Delta(\{s_i\})\}, \text{ if } n > 1 \\
\textbf{where:} & & \\
T_h(A,X) & =_{df} & A(X) \\
T_h((C \sqcap D),X) & =_{df} & T_h(C,X) \wedge T_h(D,X) \\
T_h((\forall R.C),X) & =_{df} & T_h(C,Y) \prec R(X,Y) \\
T_b(A,X) & =_{df} & A(X) \\
T_b((C \sqcap D),X) & =_{df} & T_b(C,X) \wedge T_b(D,X) \\
T_b((C \sqcup D),X) & =_{df} & T_b(C,X) \vee T_b(D,X) \\
T_b((\exists R.C),X) & =_{df} & R(X,Y) \wedge T_b(C,Y)
\end{array}
$$

Figure 1: Mapping from DL ontologies to DeLP defeasible rules

Example 1 *Consider the DL terminology $T_D = \{(b \sqsubseteq f), (c \sqsubseteq \neg f), (c \sqcap s \sqsubseteq f)\}$ that expresses both that birds fly and that chickens do not fly unless they* *are scared. The application of the $\mathcal{T}_\Delta$ mapping to T_D yields a set Δ of defeasible rules, where $\Delta = \{(f(X) \prec b(X)), (\sim f(X) \prec c(X)), (f(X) \prec c(X), s(X))\}$.*

Next we present a mapping from DL axioms to strict rules. We are going to assume that strict terminologies are consistent (see Section 5.3).

Definition 6 ($\mathcal{T}^*_\Pi$ mapping from DL sentences to DeLP strict rules) *Let A, C, D be concepts, X, Y variables, P, Q properties. The $\mathcal{T}^*_\Pi : 2^{\mathcal{L}_{DL}} \to 2^{\mathcal{L}_{DeLP_\Pi}}$ mapping is defined in Fig. 2. Besides, rules of the form "$(H_1 \wedge H_2) \leftarrow B$" are rewritten as two rules "$H_1 \leftarrow B$" and "$H_2 \leftarrow B$", rules of the form "$H_1 \leftarrow H_2 \leftarrow B$" are rewritten as "$H_1 \leftarrow B \wedge H_2$", and rules of the form "$H \leftarrow (B_1 \vee B_2)$" are rewritten as two rules "$H \leftarrow B_1$" and "$H \leftarrow B_2$".*

$$
\begin{array}{rcl}
\mathcal{T}^*_\Pi(\{C \sqsubseteq D\}) & =_{df} & \left\{\ T_h(D,X) \leftarrow T_b(C,X)\ \right\}, \\
 & & \text{if } C \text{ is an } \mathcal{L}_b\text{-class and } D \text{ an } \mathcal{L}_h\text{-class} \\
\mathcal{T}^*_\Pi(\{C \equiv D\}) & =_{df} & \mathcal{T}^*_\Pi(\{C \sqsubseteq D\}) \cup \mathcal{T}^*_\Pi(\{D \sqsubseteq C\}), \\
 & & \text{if } C \text{ and } D \text{ are } \mathcal{L}_{hb}\text{-classes} \\
\mathcal{T}^*_\Pi(\{\top \sqsubseteq \forall P.D\}) & =_{df} & \left\{\ T_h(D,Y) \leftarrow P(X,Y)\ \right\}, \\
 & & \text{if } D \text{ is an } \mathcal{L}_h\text{-class} \\
\mathcal{T}^*_\Pi(\{\top \sqsubseteq \forall P^-.D\}) & =_{df} & \left\{\ T_h(D,X) \leftarrow P(X,Y)\ \right\}, \\
 & & \text{if } D \text{ is an } \mathcal{L}_h\text{-class} \\
\mathcal{T}^*_\Pi(\{a : D\}) & =_{df} & \left\{\ T_h(D,a)\ \right\}, \\
 & & \text{if } D \text{ is an } \mathcal{L}_h\text{-class} \\
\mathcal{T}^*_\Pi(\{\langle a,b\rangle : P\}) & =_{df} & \left\{\ P(a,b)\ \right\} \\
\mathcal{T}^*_\Pi(\{P \sqsubseteq Q\}) & =_{df} & \left\{\ Q(X,Y) \leftarrow P(X,Y)\ \right\} \\
\mathcal{T}^*_\Pi(\{P \equiv Q\}) & =_{df} & \left\{\begin{array}{l} Q(X,Y) \leftarrow P(X,Y) \\ P(X,Y) \leftarrow Q(X,Y) \end{array}\right\} \\
\mathcal{T}^*_\Pi(\{P \equiv Q^-\}) & =_{df} & \left\{\begin{array}{l} Q(X,Y) \leftarrow P(Y,X) \\ P(Y,X) \leftarrow Q(X,Y) \end{array}\right\} \\
\mathcal{T}^*_\Pi(\{P^+ \sqsubseteq P\}) & =_{df} & \left\{\ P(X,Z) \leftarrow P(X,Y) \wedge P(Y,Z)\ \right\} \\
\mathcal{T}^*_\Pi(\{s_1, \dots, s_n\}) & =_{df} & \bigcup_{i=1}^{n} \mathcal{T}^*_\Pi(\{s_i\}), \text{ if } n > 1 \\
\textbf{where:} & & \\
T_h(A,X) & =_{df} & A(X) \\
T_h((C \sqcap D),X) & =_{df} & T_h(C,X) \wedge T_h(D,X) \\
T_h((\forall R.C),X) & =_{df} & T_h(C,Y) \leftarrow R(X,Y) \\
T_b(A,X) & =_{df} & A(X) \\
T_b((C \sqcap D),X) & =_{df} & T_b(C,X) \wedge T_b(D,X) \\
T_b((C \sqcup D),X) & =_{df} & T_b(C,X) \vee T_b(D,X) \\
T_b((\exists R.C),X) & =_{df} & R(X,Y) \wedge T_b(C,Y)
\end{array}
$$

Figure 2: Mapping from DL ontologies to DeLP strict rules

As DeLP is based on SLD-derivation of literals, simple translation of DL sentences to DeLP strict rules does not allow to infer negative information by *modus tollens*. For instance, "$C \sqsubseteq D$" (all C's are D's) is translated as "$D(X) \leftarrow C(X)$", DeLP is not able to derive "$\sim D(a)$" from "$\sim C(a)$". Thus given "$C_1 \sqcap C_2 \sqcap \dots \sqcap C_{n-1} \sqcap C_n \sqsubseteq D$", instead of only including the strict rule "$D(X) \leftarrow C_1(X), C_2(X), \dots, C_{n-1}(X), C_n(X)$" in its translation, we propose including all of its *transposes*.

Definition 7 (Transposes of a strict rule) *Let $r = H \leftarrow B_1, B_2, B_3, \dots, B_{n-1}, B_n$ be a DeLP strict rule. The* set of transposes of rule *r, noted as "Trans(r)", is defined as:*

$$
Trans(r) = \left\{\begin{array}{l}
H \leftarrow B_1, B_2, \dots, B_{n-1}, B_n \\
\overline{B_1} \leftarrow \overline{H}, B_2, B_3, \dots, B_{n-1}, B_n \\
\overline{B_2} \leftarrow \overline{H}, B_1, B_3, \dots, B_{n-1}, B_n \\
\overline{B_3} \leftarrow \overline{H}, B_1, B_2, \dots, B_{n-1}, B_n \\
\dots \\
\overline{B_{n-1}} \leftarrow \overline{H}, B_1, B_2, B_3 \dots, B_n \\
\overline{B_n} \leftarrow \overline{H}, B_1, B_2, \dots, B_{n-1}
\end{array}\right\}
$$

We define the mapping from DL ontologies into DeLP strict rules as $\mathcal{T}_\Pi(T) = Trans(\mathcal{T}^*_\Pi(T))$. Notice that we do not consider transposition of defeasible rules as this issue is controversial (Brewka et al. 1997, Caminada 2008).

5 DeLP-based Ontologies

As mentioned previously, traditional DL reasoners are not capable of inferring information in the presence of inconsistent ontologies. In this section, we present a framework where inconsistent definitions for concepts in an ontology are expressed are as a set of *defeasible* inclusion and equality axioms. These axioms are to be considered to be tentative. Thus, in the presence of inconsistency, to determine the epistemic status of a sentence about some individual's membership to a concept description, a dialectical analysis will be carried out considering all arguments in favor and against its membership.

5.1 Knowledge representation: δ-ontologies

An ontology is defined as a set of classes and a set of individuals belonging to such classes. We redefine the notion of DL ontology for making it suitable for our approach.

Definition 8 (δ-Ontology) *Let C be an $\mathcal{L}_b$-class, D an $\mathcal{L}_h$-class, A, B $\mathcal{L}_{hb}$-classes, P, Q properties, a, b individuals. Let T be a set of inclusion and equality sentences in $\mathcal{L}_{DL}$ of the form $C \sqsubseteq D$, $A \equiv B$, $\top \sqsubseteq \forall P.D$, $\top \sqsubseteq \forall P^-.D$, $P \sqsubseteq Q$, $P \equiv Q$, $P \equiv Q^-$, o $P^+ \sqsubseteq P$ such that T can be partitioned into two disjoint sets T_S and T_D. Let A be a set of assertions disjoint with T of the form $a : D$ or $\langle a,b\rangle : P$. A δ-ontology Σ is a tuple (T_S, T_D, A). The set T_S is called the* strict terminology *(or Sbox), T_D the* defeasible terminology *(or Dbox) and A the* assertional box *(or Abox).*

Example 2 *Let $\Sigma_2 = (T_S, T_D, A)$ be a δ-ontology, where:*[2] *$T_S = \{(c \sqcup p \sqsubseteq b), (p \sqsubseteq \neg f)\}$; $T_D = \{(b \sqsubseteq f), (c \sqsubseteq \neg f), (c \sqcap s \sqsubseteq f), (f \sqsubseteq nit)\}$, and $A = \{(ti : c), (tw : p), (ti : s)\}$. The Sbox T_S says both that chickens and penguins are birds, and that penguins do not fly. The Dbox T_D expresses that birds usually fly, chickens typically do not fly unless they are scared, and that flying animals normally nest in trees. The Abox A establishes that Tina is a chicken, Tweety is a penguin and Tina is scared.*

We will see how to assign semantics to δ-ontologies as DeLP programs. The Sbox will be interpreted as a set of strict rules, the Abox as set of facts and the Dbox as a set of defeasible rules.

5.2 Semantic interpretation of δ-ontologies as DeLP programs

The traditional approach to reasoning in DLs is based on a model-theoretic semantics. As DLs are a subset of first-order logic (FOL), entailment has an *explosive* effect in the presence of inconsistent ontologies. In this work, we propose an argumentative approach to reasoning with inconsistent ontologies. Thus a δ-ontology will be interpreted as a DeLP program. As required by the DeLP framework, we are assuming that under a traditional interpretation the set $T_S \cup A$ has a model (see Section 5.3).

Definition 9 (Interpretation of a δ-ontology) *Let $\Sigma = (T_S, T_D, A)$ be a δ-ontology. The* interpretation of Σ *is a DeLP program $\mathcal{P} = (\mathcal{T}_\Pi(T_S) \cup \mathcal{T}_\Pi(A), \mathcal{T}_\Delta(T_D))$.*

Example 3 (Continues Ex. 2) *Consider again the δ-ontology Σ_2. Σ_2 is interpreted as the DeLP*

[2]This example appears in (García & Simari 2004) in a different context.

program $\mathcal{P}_2 = (\Pi, \Delta)$, where: $\Pi = \{(b(X) \leftarrow c(X)), (\sim c(X) \leftarrow\sim b(X)), (b(X) \leftarrow p(X)), (\sim p(X) \leftarrow\sim b(X)), (\sim f(X) \leftarrow p(X)), (\sim p(X) \leftarrow f(X)), c(ti), p(tw), s(ti)\}$, and $\Delta = \{ (f(X) \prec b(X)), (\sim f(X) \prec c(X)), (f(X) \prec c(X), s(X)), (nit(X) \prec f(X))\}$.

5.3 Inference tasks in δ-ontologies

In the DL approach to reasoning with ontologies in the Semantic Web, once a knowledge engineer has designed the terminology and used the DL reasoning service for checking that all of the terminology's concepts are satisfiable, the Abox can be filled with assertions about individuals. In order to keep consistency within an argument as required by Def. 2, we must enforce some internal coherence between the Abox and the Tbox. Formally:

Definition 10 (Internal coherence in Aboxes. Consistency of Aboxes w.r.t. Sboxes) *Let $\Sigma = (T_S, T_D, A)$ be a δ-ontology. Let C be a class name, a, b individuals. The Abox A is* internally coherent *iff there are no pair of assertions $a : C$ and $a : \neg C$. The Abox A is* consistent w.r.t. *the terminology T_S iff it is not possible to derive two literals $C(a)$ and $\sim C(a)$ from $\mathcal{T}_\Pi(T_S) \cup \mathcal{T}_\Pi(A)$.*

Example 4 *Let $\Sigma_4 = (T_S, \emptyset, A)$ be a δ-ontology such that $T_S = \{(C \sqsubseteq D), (D \sqsubseteq \neg F)\}$ and $A = \{(a : C), (a : F)\}$. Σ_4 is expressed as $\mathcal{P}_4 = (\Pi, \emptyset)$, where $\Pi = \mathcal{T}_\Pi(T_S) \cup \mathcal{T}_\Pi(A) = \{C(a), F(a), (D(X) \leftarrow C(X)), (\sim C(X) \leftarrow\sim D(X)), (\sim F(X) \leftarrow D(X)), (\sim D(X) \leftarrow F(X))\}$, from which is possible to strictly derive $F(a)$ and $\sim F(a)$. Therefore A is not coherent w.r.t. T_S.*

5.3.1 Instance checking

In the traditional DL setting, *instance checking* refers to determining whether the assertions in the Abox entail that a particular individual is an instance of a given concept description (Baader et al. 2003). We propose a set of definitions to capture this notion in the context of δ-ontologies.

Definition 11 (Potential, justified and strict membership of an individual to a class) *Let $\Sigma = (T_S, T_D, A)$ be a δ-ontology. Let C be a class name, a an individual, let $\mathcal{P} = (\mathcal{T}_\Pi(T_S) \cup \mathcal{T}_\Pi(A), \mathcal{T}_\Delta(T_D))$. (a) The* individual a potentially belongs to class C *(noted as "C_p^a") iff there exists an argument $\langle \mathcal{A}, C(a)\rangle$ w.r.t. $\mathcal{P}$; (b) the* individual a justifiedly belongs to class C *(noted as "C_j^a") iff there exists a warranted argument $\langle \mathcal{A}, C(a)\rangle$ w.r.t. $\mathcal{P}$, and, (c) the* individual a strictly belongs to class C *(noted as "C_s^a") iff there exists an argument $\langle \emptyset, C(a)\rangle$ w.r.t. $\mathcal{P}$.*

Property 1 *C_s^a implies C_j^a, and C_j^a implies C_p^a.*

Proof: The former holds because in DeLP empty arguments (*i.e.*, literals derived exclusively from strict rules) have no defeaters and they are thus warranted. The latter trivially holds because warranted arguments are arguments.

We now extend the notion of membership to arbitrary concept expressions.

Definition 12 (Potential, justified and strict membership of an individual to a class (extended version)) *Let $\Sigma = (T_S, T_D, A)$ be a δ-ontology. Let C, D class names in Σ, a, b individuals in Σ, R an atomic property in Σ, $\mathcal{P} = (\mathcal{T}_\Pi(T_S) \cup \mathcal{T}_\Pi(A), \mathcal{T}_\Delta(T_D))$. Let E be a class name not present in Σ. The* potential (justified resp.) membership *of a to a complex concept is defined as:*

- *$\neg C$: $(\neg C)_p^a$ ($(\neg C)_j^a$, resp.) iff there exists an argument (warranted argument, resp.) $\langle \mathcal{A}, \sim C(a)\rangle$ w.r.t. $\mathcal{P}$.*
- *$C \sqcap D$: Let $\mathcal{P}_2 = (\Pi, \Delta \cup \mathcal{T}_\Delta(C \sqcap D \sqsubseteq E))$. $(C \sqcap D)_p^a$ ($(C \sqcap D)_j^a$, resp.) iff there exists an argument (warranted argument, resp.) $\langle \mathcal{A}, E(a)\rangle$ w.r.t. $\mathcal{P}_2$.*
- *$C \sqcup D$: Let $\mathcal{P}_2 = (\Pi, \Delta \cup \mathcal{T}_\Delta(C \sqcup D \sqsubseteq E))$. $(C \sqcup D)_p^a$ ($(C \sqcup D)_j^a$, resp.) iff there exists an argument (warranted argument, resp.) $\langle \mathcal{A}, E(a)\rangle$ w.r.t. $\mathcal{P}_2$.*
- *$\exists R.C$: Let $\mathcal{P}_2 = (\Pi, \Delta \cup \mathcal{T}_\Delta(\exists R.C \sqsubseteq E))$. $(\exists R.C)_p^a$ ($(\exists R.C)_j^a$, resp.) iff there exists an argument (warranted argument, resp.) $\langle \mathcal{A}, E(a)\rangle$ w.r.t. $\mathcal{P}_2$.*

The strict membership *of a to a complex concept is defined as:*

- *$\neg C$: $(\neg C)_s^a$ iff there exists an argument $\langle \emptyset, \sim C(a)\rangle$ w.r.t. $\mathcal{P}$.*
- *$C \sqcap D$: Let $\mathcal{P}_2 = (\Pi \cup \mathcal{T}_\Pi(C \sqcap D \sqsubseteq E), \Delta)$. $(C \sqcap D)_s^a$ iff there exists an argument $\langle \emptyset, E(a)\rangle$ w.r.t. $\mathcal{P}_2$.*
- *$C \sqcup D$: Let $\mathcal{P}_2 = (\Pi \cup \mathcal{T}_\Pi(C \sqcup D \sqsubseteq E), \Delta)$. $(C \sqcup D)_s^a$ iff there exists an argument $\langle \emptyset, E(a)\rangle$ w.r.t. $\mathcal{P}_2$.*
- *$\exists R.C$: Let $\mathcal{P}_2 = (\Pi \cup \mathcal{T}_\Pi(\exists R.C \sqsubseteq E), \Delta)$. $(\exists R.C)_s^a$ iff there exists an argument $\langle \emptyset, E(a)\rangle$ w.r.t. $\mathcal{P}_2$.*

Property 2 *Let $\Sigma = (T_S, T_D, A)$ be a δ-ontology. It cannot be the case that individual a belongs justifiedly to concept C and $\neg C$ simultaneously.*

Proof: Suppose that both C_j^a and $(\neg C)_j^a$. Then it must be the case there exist two warranted arguments $\langle \mathcal{A}, C(a)\rangle$ and $\langle \mathcal{B}, \sim C(a)\rangle$ w.r.t. $(\mathcal{T}_\Pi(T_S) \cup \mathcal{T}_\Pi(A), \mathcal{T}_\Delta(T_D))$. But this impossible as DeLP cannot warrant two complementary literals simultaneously (as proven by García & Simari (2004)).

Property 3 *Let $\Sigma = (T_S, T_D, A)$ be a δ-ontology. It cannot be the case that individual a belongs strictly to concept C and $\neg C$ simultaneously.*

Proof: If C_s^a and $(\neg C)_s^a$, then there must exist two arguments $\langle \emptyset, C(a)\rangle$ and $\langle \emptyset, \sim C(a)\rangle$ w.r.t. $(\mathcal{T}_\Pi(T_S) \cup \mathcal{T}_\Pi(A), \mathcal{T}_\Delta(T_D))$. Then $\mathcal{T}_\Pi(T_S) \cup \mathcal{T}_\Pi(A)$ is inconsistent (contradicting Def. 10).

5.3.2 Retrieval

When DL knowledge bases are considered, we would want to know all individuals that are instances of a certain concept. In the traditional DL setting, given an ontology (T, A) and a concept C, in the *retrieval problem* we are interested to know all of the individuals a such that $T \cup A \models a : C$ (Baader et al. 2003). We present naïve solutions to two related problems:

- *Open retrieval:* Given a δ-ontology $\Sigma = (T_S, T_D, A)$ and a class C, find all individuals which are instances of C. We solve this problem by finding all the individuals a such that there exists a warranted argument $\langle \mathcal{A}, C(a)\rangle$ w.r.t. DeLP program $(\mathcal{T}_\Pi(T_S) \cup \mathcal{T}_\Pi(A), \mathcal{T}_\Delta(T_D))$.
- *Retrieval of all classes:* Given a δ-ontology $\Sigma = (T_S, T_D, A)$ and an individual a, find all named classes C such that a is an instance of C. We solve this problem by finding all the classes C such that there exists a warranted argument $\langle \mathcal{A}, C(a)\rangle$ w.r.t. DeLP program $(\mathcal{T}_\Pi(T_S) \cup \mathcal{T}_\Pi(A), \mathcal{T}_\Delta(T_D))$.

Property 4 *The running time of processes for "open retrieval" and for "retrieval of all classes" is finite.*

Proof: As a δ-ontology has a finite number of both named concepts and individual constants, the DeLP program obtained from it is finite. Cecchi et al. (2006) have shown that determining if there exists an argument for a literal is NP; besides, as the warrant procedure always builds a finite dialectical tree (García & Simari 2004), then proceses for "open retrieval" and for "retrieval of all classes" always terminate.

6 DeLP-based Integration of DL Ontologies

In this section, we introduce an application for ontology integration in the Semantic Web based on the framework presented above. *Ontology integration* is the problem of combining ontologies residing in different sources and to provide the user with an unified view of such ontologies (Calvanese et al. 2001). The problem of designing systems for ontology integration in the Semantic Web is particularly important because ontologies are to be developed independently from each other and, for this reason, they can be mutually inconsistent. One possible architecture for ontology integration systems is based on a global schema and a set of local sources. The local sources contain the actual data while the global schema provides a reconciled, unified view of the underlying sources. A basic service provided by ontology integration systems is that of answering queries posed in terms of the global schema.

Definition 13 (Ontology integration system) *An* ontology integration system $\mathcal{I}$ *is a triple* $(\mathcal{G}, \mathcal{S}, \mathcal{M})$ *where:*

- $\mathcal{G}$ *is a* global ontology *expressed as a* δ*-ontology over an alphabet* $\mathcal{A}_\mathcal{G}$.
- $\mathcal{S}$ *is a* set of n source ontologies $\mathcal{S}_1, \ldots, \mathcal{S}_n$ *expressed as* δ*-ontologies over alphabets* $\mathcal{A}_{\mathcal{S}_1}, \ldots, \mathcal{A}_{\mathcal{S}_n}$*, resp. Each alphabet* $\mathcal{A}_{\mathcal{S}_i}$ *includes a symbol for each element of the source* $\mathcal{S}_i$*,* $i = 1, \ldots, n$.
- $\mathcal{M}$ *is a* set of n mappings $\mathcal{M}_1, \ldots, \mathcal{M}_n$ *between* $\mathcal{G}$ *and* $\mathcal{S}_1, \ldots, \mathcal{S}_n$*, resp. Each mapping* $\mathcal{M}_i$ *is constituted by a set of* assertions *of the form* $q_{\mathcal{S}_i} \sqsubseteq q_\mathcal{G}$*, where* $q_\mathcal{G}$ *y* $q_{\mathcal{S}_i}$ *are queries of the same arity defined over global ontology* $\mathcal{G}$ *and* $\mathcal{S}_i$*,* $i = 1, \ldots, n$*, resp. Queries* $q_\mathcal{G}$ *are expressed over alphabet* $\mathcal{A}_\mathcal{G}$ *and queries* $q_{\mathcal{S}_i}$ *are expressed over alphabet* $\mathcal{A}_{\mathcal{S}_i}$*. The sets* $\mathcal{M}_1, \ldots, \mathcal{M}_n$ *are called* bridge ontologies.

Next we show a case study in which several DL *local* ontologies and a DL *global* ontology are integrated. These ontologies will be interpreted as DeLP programs. Queries posed w.r.t. the global ontology are going to be interpreted as queries answered on the basis of such DeLP programs.

Example 5 *Consider the* global *δ-ontology* $\mathcal{G} = (\emptyset, T_D^\mathcal{G}, \emptyset)$ *presented in Fig. 3. The Dbox* $T_D^\mathcal{G}$ *expresses that computer geeks are usually not good in sports; expert swimmers are normally good at sports, and, if somebody is either capable of swimming both a race stroke and a rescue stroke or is a diver, then she is typically considered an expert swimmer.*

Defeasible terminology $T_D^\mathcal{G}$**:**
$geek \sqsubseteq \neg good$
$swimmer \sqsubseteq good$
$(\exists can\_swim.rescue\_stroke \sqcap \exists can\_swim.race\_stroke)$
$\sqcup\ diver \sqsubseteq swimmer$

Figure 3: Global ontology $\mathcal{G} = (\emptyset, T_D^\mathcal{G}, \emptyset)$

Notice that the terminology $T_D^\mathcal{G}$ expresses a conflict w.r.t. concept "*good*". If we find that some individual in the Abox can be proven to be member of both concepts "*swimmer*" and "*geek*", then the Abox would be incoherent from a traditional DL point of view because that individual would belong both to "*good*" and to "$\neg$*good*, indicating that concept "*good*" should be empty and having one individual at the same time. We will show how this situation can be handled naturally in DeLP.

Example 6 (Continues Ex. 5) *In Fig. 4, we present two source ontologies:* $\mathcal{S}_1$ *about water activities, and* $\mathcal{S}_2$ *on computer programming. In source local ontology* $\mathcal{S}_1 = (\emptyset, T_D^{\mathcal{S}_1}, A^{\mathcal{S}_1})$*, Dbox* $T_D^{\mathcal{S}_1}$ *says that both free and scuba divers are divers, saturation divers are a particular class of scuba divers, and somebody capable of swimming some kind of stroke is usually a swimmer. Abox* $A^{\mathcal{S}_1}$ *establishes that John swims both crawl and side strokes, Paul is a saturation diver, and crawl and side are swimming strokes.*

In source local ontology $\mathcal{S}_2 = (T_S^{\mathcal{S}_2}, T_D^{\mathcal{S}_2}, A^{\mathcal{S}_2})$*, Sbox* $T_S^{\mathcal{S}_2}$ *expresses that among programming languages, both logic programming and object-oriented languages can be found. Dbox* $T_D^{\mathcal{S}_2}$ *says that a programmer is usually somebody who can program in some programming language, and that someone who can read and write code in such a language can program unless she has failed the elementary programming course. Abox* $A^{\mathcal{S}_2}$ *establishes that Prolog is a logic programming language and that John can read and write Prolog code; that Java is an object-oriented language and that Mary can read and write Java code, and that Paul is capable of reading and writing Java code although he failed the elementary programming course.*

Source ontology $\mathcal{S}_1 = (\emptyset, T_D^{\mathcal{S}_1}, A^{\mathcal{S}_1})$**:**

Defeasible terminology $T_D^{\mathcal{S}_1}$**:**
$free\_diver \sqcup scuba\_diver \sqsubseteq diver$
$saturation\_diver \sqsubseteq scuba\_diver$
$\exists swims.stroke \sqsubseteq swimmer$

Assertional box $A^{\mathcal{S}_1}$**:**

$crawl : stroke$;	$side : stroke$
$\langle john, crawl\rangle : swims$;	$\langle john, side\rangle : swims$
$paul : saturation\_diver$	

Source ontology $\mathcal{S}_2 = (T_S^{\mathcal{S}_2}, T_D^{\mathcal{S}_2}, A^{\mathcal{S}_2})$**:**

Strict terminology $T_S^{\mathcal{S}_2}$**:**
$lp\_lang \sqcup oop\_lang \sqsubseteq lang$

Defeasible terminology $T_D^{\mathcal{S}_2}$**:**
$\exists programs.lang \sqsubseteq programmer$
$\exists programs.lang \sqcap failed\_prog\_101 \sqsubseteq \neg programmer$
$reads \sqcap writes \sqsubseteq programs$

Assertional box $A^{\mathcal{S}_2}$**:**

$prolog : lp\_lang$;	$java : oop\_lang$
$\langle john, prolog\rangle : reads$;	$\langle john, prolog\rangle : writes$
$\langle mary, java\rangle : reads$;	$\langle mary, java\rangle : writes$
$\langle paul, java\rangle : reads$;	$\langle paul, java\rangle : writes$
$paul : failed\_prog\_101$	

Figure 4: Source ontologies $\mathcal{S}_1$ and $\mathcal{S}_2$

Notice how, in particular, source ontology $\mathcal{S}_2$ is inconsistent from a traditional point of view because the individual named Paul belongs at the same time to concepts "*programmer*" and "$\neg$*programmer*". Therefore, this ontology cannot be processed by traditional DL reasoners. We will show how can this be achieved in the framework of δ-ontologies.

As mentioned above, our goal is to answer queries about the membership of individuals to a certain concept of a global ontology using the data defined in source ontologies. The relationship of the global ontology with the local ontologies is achieved through *bridge ontologies*. A bridge ontology allows to map concepts and properties between two ontologies. Thus a concept in one ontology corresponds to a *view* of one or several concepts in some other ontology. In the examples we present, we consider bridge ontologies as *given*; for techniques on semi-automatic discovery of such mappings implemented as articulation rules, see (Mitra 2004). Moreover, we are going to assume *unique name assumption* w.r.t. references to individuals in Aboxes through our presentation.

Example 7 (Continues Ex. 6) *Consider again global ontology $\mathcal{G}$ and source ontologies $\mathcal{S}_1$ and $\mathcal{S}_2$. Definitions in $\mathcal{G}$ with those in $\mathcal{S}_1$ and $\mathcal{S}_2$ are articulated by bridge ontologies $\mathcal{M}_1$ and $\mathcal{M}_2$, resp. (see Fig. 5). For clarity, in bridge ontologies we qualify concept and property names with their defining ontology name.*

Bridge ontology $\mathcal{M}_1$ expresses that concept "swims" in $\mathcal{S}_1$ corresponds to concept "can_swim" in $\mathcal{G}$; concept "diver" in $\mathcal{S}_1$ refers to "diver" in $\mathcal{G}$; concept (anonimously defined by the one-of *construct) composed of individual "side" in $\mathcal{S}_1$ is mapped to the concept "rescue_stroke" in $\mathcal{G}$, and, the concept composed by individual "crawl" in $\mathcal{S}_1$ is mapped to "race_stroke" in $\mathcal{G}$. Bridge ontology $\mathcal{M}_2$ indicates that concept "programmer" in $\mathcal{S}_2$ corresponds to concept "geek" defined in $\mathcal{G}$.*

Bridge ontology $\mathcal{M}_1$ between $\mathcal{G}$ and $\mathcal{S}_1$:

$\mathcal{S}_1 : swims \sqsubseteq \mathcal{G} : can\_swim$
$\mathcal{S}_1 : diver \sqsubseteq \mathcal{G} : diver$
$\mathcal{S}_1 : \{side\} \sqsubseteq \mathcal{G} : rescue\_stroke$
$\mathcal{S}_1 : \{crawl\} \sqsubseteq \mathcal{G} : race\_stroke$

Bridge ontology $\mathcal{M}_2$ between $\mathcal{G}$ and $\mathcal{S}_2$:

$\mathcal{S}_2 : programmer \sqsubseteq \mathcal{G} : geek$

Figure 5: Bridge ontologies $\mathcal{M}_1$ and $\mathcal{M}_2$

As discussed above, in the *global-as-view* approach to ontology integration, queries are posed w.r.t. a global ontology which is used as a means to access data found in local source ontologies. Next we show how to extend the task of instance checking for individual membership to concepts defined in a global ontology in the context of an ontology integration system. We will show how an ontology integration system can be regarded as a DeLP program and queries to the ontology integration system can be interpreted as queries w.r.t. such DeLP program.

Definition 14 (Interpretation of an ontology integration system) *Let $\mathcal{I} = (\mathcal{G}, \mathcal{S}, \mathcal{M})$ be an ontology integration system such that $\mathcal{S} = \{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$ y $\mathcal{M} = \{\mathcal{M}_1, \ldots, \mathcal{M}_n\}$, where:*

- $\mathcal{G} = (T_S^{\mathcal{G}}, T_D^{\mathcal{G}}, A^{\mathcal{G}})$*;*
- $\mathcal{S}_i = (T_S^{\mathcal{S}_i}, T_D^{\mathcal{S}_i}, A_i^{\mathcal{S}_i})$*, with $i = 1, \ldots, n$, and,*
- $\mathcal{M}_i = (T_S^{\mathcal{M}_i}, T_D^{\mathcal{M}_i})$*, with $i = 1, \ldots, n$.*

The system $\mathcal{I}$ is interpreted as the DeLP program $\mathcal{I}_{DeLP} = (\Pi, \Delta)$:

$$\Pi = \left(\mathcal{T}_\Pi(T_S^{\mathcal{G}}) \right) \cup \left(\mathcal{T}_\Pi(A^{\mathcal{G}}) \right) \cup \left(\bigcup_{i=1}^{n} \mathcal{T}_\Pi(T_S^{\mathcal{S}_i}) \right) \cup \left(\bigcup_{i=1}^{n} \mathcal{T}_\Pi(T_S^{\mathcal{M}_i}) \right);$$

$$\Delta = \left(\mathcal{T}_\Delta(T_D^{\mathcal{G}}) \right) \cup \left(\bigcup_{i=1}^{n} \mathcal{T}_\Delta(T_D^{\mathcal{S}_i}) \right) \cup \left(\bigcup_{i=1}^{n} \mathcal{T}_\Delta(T_D^{\mathcal{M}_i}) \right).$$

Example 8 (Continues Ex. 7) *The interpretation as DeLP programs of global ontology $\mathcal{G}$; sources $\mathcal{S}_1$ and $\mathcal{S}_2$, and bridges $\mathcal{M}_1$ and $\mathcal{M}_2$ are shown in Figs. 6, 7 and 8, resp. Global ontology $\mathcal{G}$ is interpreted as the DeLP program $\mathcal{P}_{\mathcal{G}} = (\emptyset, \Delta_{\mathcal{G}})$; source ontology $\mathcal{S}_1$, as $\mathcal{P}_1 = (\Pi_1, \Delta_1)$; source ontology $\mathcal{S}_2$, as $\mathcal{P}_2 = (\Pi_2, \Delta_2)$; bridge ontology $\mathcal{M}_1$, as the set of defeasible rules $\Delta_{\mathcal{M}_1}$, and, bridge ontology $\mathcal{M}_2$, as $\Delta_{\mathcal{M}_2}$.*

Thus, the interpretation of $\mathcal{I}$ is the DeLP program $\mathcal{P}_{\mathcal{I}} = (\Pi, \Delta)$ where $\Pi = \Pi_1 \cup \Pi_2$, and $\Delta = \Delta_{\mathcal{G}} \cup \Delta_1 \cup \Delta_2 \cup \Delta_{\mathcal{M}_1} \cup \Delta_{\mathcal{M}_2}$.

DeLP program $\mathcal{P}_{\mathcal{G}} = (\emptyset, \Delta_{\mathcal{G}})$ obtained from $\mathcal{G}$:

Defeasible rules $\Delta_{\mathcal{G}}$:

$\sim good(X) \prec geek(X)$
$good(X) \prec swimmer(X)$
$swimmer(X) \prec$
$\quad can\_swim(X, Y), rescue\_stroke(Y),$
$\quad can\_swim(X, Z), race\_stroke(Z)$
$swimmer(X) \prec diver(X)$

Figure 6: DeLP program $\mathcal{P}_{\mathcal{G}} = (\emptyset, \Delta_{\mathcal{G}})$ obtained from global ontology $\mathcal{G}$

DeLP program $\mathcal{P}_1 = (\Pi_1, \Delta_1)$ obtained from $\mathcal{S}_1$:

Facts Π_1:

$stroke(crawl)$;	$stroke(side)$
$swims(john, crawl)$;	$swims(john, side)$
$saturation\_diver(paul)$	

Defeasible rules Δ_1:

$diver(X) \prec free\_diver(X)$
$diver(X) \prec scuba\_diver(X)$
$scuba\_diver(X) \prec saturation\_diver(X)$
$swimmer(X) \prec swims(X, Y), stroke(Y)$

DeLP program $\mathcal{P}_2 = (\Pi_2, \Delta_2)$ obtained from $\mathcal{S}_2$:

Facts and strict rules Π_2:

$lp\_lang(prolog)$;	$oop\_lang(java)$
$reads(john, prolog)$;	$writes(john, prolog)$
$reads(mary, java)$;	$writes(mary, java)$
$reads(paul, java)$;	$writes(paul, java)$
$failed\_prog\_101(paul)$	

$lang(X) \leftarrow lp\_lang(X)$
$\sim lp\_lang(X) \leftarrow \sim lang(X)$
$lang(X) \leftarrow oop\_lang(X)$
$\sim oop\_lang(X) \leftarrow \sim lang(X)$

Defeasible rules Δ_2:

$programmer(X) \prec programs(X, Y), lang(Y)$
$\sim programmer(X) \prec$
$\quad programs(X, Y), lang(Y), failed\_prog\_101(X)$
$programs(X, Y) \prec reads(X, Y), writes(X, Y)$

Figure 7: DeLP programs $\mathcal{P}_1$ and $\mathcal{P}_2$ obtained from source ontologies $\mathcal{S}_1$ and $\mathcal{S}_2$, resp.

Possible inferences in the integrated ontology $\mathcal{I}_{DeLP}$ are modeled by means of a dialectical analysis in the DeLP program that is obtained when each DL sentence of the ontology is mapped into DeLP clauses. Thus warranted arguments will be the valid consequences that will be obtained from the original ontology, provided the strict information in $\mathcal{I}_{DeLP}$ is consistent. Formally:

Definition 15 (Potential, justified and strict membership of individuals to concepts in ontology integration systems) *Let $\mathcal{I} = (\mathcal{G}, \mathcal{S}, \mathcal{M})$ be an ontology integration system. Let a be an individual name, and c a concept name defined in $\mathcal{G}$. Individual* a belongs potentially *to concept C iff there exists an argument $\mathcal{A}$ for the literal $C(a)$ w.r.t. DeLP program $\mathcal{I}_{DeLP}$. Individual* a belongs justifiedly *to concept C iff there exists a warranted argument $\mathcal{A}$ for the literal*

Bridge rules $\Delta_{\mathcal{M}_1}$ between ontologies $\mathcal{G}$ and $\mathcal{S}_1$:

$\mathcal{G} : can\_swim(X, Y) \prec \mathcal{S}_1 : swims(X, Y)$
$\mathcal{G} : diver(X) \prec \mathcal{S}_1 : diver(X)$
$\mathcal{G} : rescue\_stroke(X) \prec \mathcal{S}_1 : stroke(X), X = side$
$\mathcal{G} : race\_stroke(X) \prec \mathcal{S}_1 : stroke(X), X = crawl$

Bridge rules $\Delta_{\mathcal{M}_2}$ between ontologies $\mathcal{G}$ and $\mathcal{S}_2$:

$\mathcal{G} : geek(X) \prec \mathcal{S}_2 : programmer(X)$

Figure 8: Bridge ontologies expressed as defeasible rules

$C(a)$ w.r.t. DeLP program $\mathcal{I}_{DeLP}$. Individual a belongs strictly to concept C iff there exists an empty argument for the literal $C(a)$ w.r.t. DeLP program $\mathcal{I}_{DeLP}$.

Next we will show some of the arguments that can be built from the integrated ontology system. In the rest of the presentation, we are assuming generalized specificity (Simari & Loui 1992) as the criterion for argument comparison.

Example 9 (Continues Ex. 8) *Consider again DeLP program $\mathcal{I}_{DeLP}$, we are interested in determining the justified membership of individuals John, Mary and Paul to concepts "good" and/or "¬good". According to Def. 15, it is necessary to determine if there exist warranted arguments for literals good(john), good(mary) and good(paul), resp. Notice that answers to queries cannot be ambiguous as it is not possible to warrant complementary literals in DeLP at the same time (see Section 7). We will see that as John is both a geek and a swimmer, it will not be possible to determine if he is or not good at sports. In spite of this result, we will also see that as Mary is a Java programmer, she will not be regarded as good at sports. In the case of Paul, as he is a diver and, although he programs in Java but failed the elementary programming course, he will not be considered a programmer and thus, he will be regarded as good at sports.*

First, we will consider the dialectical analysis for the query "good(john)". There are reasons to assert that John belongs potentially to concept "good". Formally, there exists an argument $\langle \mathcal{A}_1, good(john)\rangle$ where:

$$\mathcal{A}_1 = \left\{ \begin{array}{l} (good(john) \mathrel{-\!\!\prec} swimmer(john)), \\ (swimmer(john) \mathrel{-\!\!\prec} \\ \quad can\_swim(john, side), rescue\_stroke(side), \\ \quad can\_swim(john, crawl), race\_stroke(crawl)), \\ (\mathcal{G} : race\_stroke(crawl) \mathrel{-\!\!\prec} \\ \quad \mathcal{S}_1 : stroke(crawl), crawl = crawl), \\ (\mathcal{G} : can\_swim(john, crawl) \mathrel{-\!\!\prec} \\ \quad \mathcal{S}_1 : swims(john, crawl)), \\ (\mathcal{G} : rescue\_stroke(side) \mathrel{-\!\!\prec} \\ \quad \mathcal{S}_1 : stroke(side), side = side) \\ (\mathcal{G} : can\_swim(john, side) \mathrel{-\!\!\prec} \\ \quad \mathcal{S}_1 : swims(john, side)) \end{array} \right\}$$

However, John belongs potentially to concept "¬good" because he is a computer geek. Formally, there is an argument $\langle \mathcal{A}_2, \sim good(john)\rangle$ that defeats argument $\mathcal{A}_1$, where:

$$\mathcal{A}_2 = \left\{ \begin{array}{l} (\sim good(john) \mathrel{-\!\!\prec} geek(john)), \\ (\mathcal{G} : geek(john) \mathrel{-\!\!\prec} \mathcal{S}_2 : programmer(john)), \\ (programmer(john) \mathrel{-\!\!\prec} \\ \quad programs(john, prolog), lang(prolog)), \\ (programs(john, prolog) \mathrel{-\!\!\prec} \\ \quad reads(john, prolog), writes(john, prolog)) \end{array} \right\}.$$

Thus, in the dialectical tree for the query "good(john)", defeated argument $\mathcal{A}_1$ appears labeled as a D-node while victorious argument $\mathcal{A}_2$ appears marked as a U-node. (see Fig. 9.(a)). On the other hand, when we consider the membership of John to concept "¬good", we discover that argument $\mathcal{A}_2$ supporting this conclusion is defeated by argument $\mathcal{A}_1$ (see Fig. 9.(b)). Therefore, the answer to query "good(john)" is UNDECIDED.

Second, we consider the dialectical analysis for determining if Mary belongs to concept "good". Mary belongs justifiedly to concept "¬good" as the answer to query "good(mary)" is NO *because there is a warranted argument $\langle \mathcal{B}, \sim good(mary)\rangle$ (see Fig. 9.(c)), where:*

$$\mathcal{B} = \left\{ \begin{array}{l} (\sim good(mary) \mathrel{-\!\!\prec} geek(mary)), \\ (\mathcal{G} : geek(mary) \mathrel{-\!\!\prec} \mathcal{S}_2 : programmer(mary)), \\ (programmer(mary) \mathrel{-\!\!\prec} \\ \quad programs(mary, java), lang(java)), \\ (lang(java) \mathrel{-\!\!\prec} oop\_lang(java)), \\ (programs(mary, java) \mathrel{-\!\!\prec} \\ \quad reads(mary, java), writes(mary, java)) \end{array} \right\}$$

Third, we will see why Paul belongs justifiedly to concept "good". Let us consider the dialectical tree for the literal "good(paul)". There is an argument $\langle \mathcal{C}_1, good(paul)\rangle$, based on the defeasible information that expresses that Paul is an expert swimmer (because he is a saturation diver):

$$\mathcal{C}_1 = \left\{ \begin{array}{l} (good(paul) \mathrel{-\!\!\prec} swimmer(paul)), \\ (swimmer(paul) \mathrel{-\!\!\prec} \mathcal{G} : diver(paul)), \\ (\mathcal{G} : diver(paul) \mathrel{-\!\!\prec} \mathcal{S}_1 : diver(paul)), \\ (\mathcal{S}_1 : diver(paul) \mathrel{-\!\!\prec} scuba\_diver(paul)), \\ (scuba\_diver(paul) \mathrel{-\!\!\prec} saturation\_diver(paul)) \end{array} \right\}$$

But argument $\mathcal{C}_1$ is attacked by an argument $\langle \mathcal{C}_2, \sim good(paul)\rangle$, where:

$$\mathcal{C}_2 = \left\{ \begin{array}{l} (\sim good(paul) \mathrel{-\!\!\prec} geek(paul)), \\ (\mathcal{G} : geek(paul) \mathrel{-\!\!\prec} \mathcal{S}_2 : programmer(paul)), \\ (programmer(paul) \mathrel{-\!\!\prec} \\ \quad programs(paul, java), lang(java)), \\ (programs(paul, java) \mathrel{-\!\!\prec} \\ \quad reads(paul, java), writes(paul, java)) \end{array} \right\}$$

Nevertheless, Paul also belongs potentially to concept "$\mathcal{S}_2$: ¬programmer" (because he failed the elementary programming course), as an argument $\langle \mathcal{C}_3, \mathcal{S}_2 : \sim programmer(paul)\rangle$ can be found, where:

$$\mathcal{C}_3 = \left\{ \begin{array}{l} (\sim programmer(paul) \mathrel{-\!\!\prec} \\ \quad programs(paul, java), lang(java), \\ \quad failed\_prog\_101(paul)), \\ (programs(paul, java) \mathrel{-\!\!\prec} \\ \quad reads(paul, java), writes(paul, java)) \end{array} \right\}$$

Thus, the dialectical tree for the query "good(paul)" has three nodes (see Fig. 9.(d)). Respect the query "∼good(paul)" for determining if Paul belongs justifiedly to concept "¬good", argument $\mathcal{C}_2$ is defeated by argument $\mathcal{C}_1$ (see Fig. 9.(e)). Therefore, the answer to the query "good(paul)" is YES, *and we conclude that Paul belongs justifiedly to concept "good".*

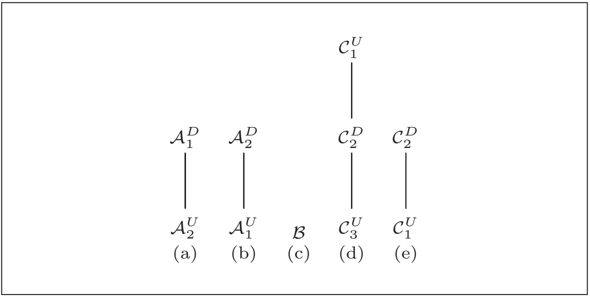

Figure 9: Dialectical trees for queries *good*(*john*), *good*(*mary*) and *good*(*paul*)

7 Evaluation of the Proposal

In order to evaluate our approach, we propose using the framework presented by Huang et al. (2005) for reasoning with inconsistent ontologies with a non-standard inference relation. With classical reasoning, a query ϕ given an ontology Σ can be expressed as an evaluation of the consequence relation $\Sigma \models \phi$; there are two answers to a query: either "yes" ($\Sigma \models \phi$) or "no" ($\Sigma \not\models \phi$). For reasoning with inconsistent ontologies with a non-standard inference relation, Huang et al. (2005) propose using an alternative classification to distinguish answers to queries:

Definition 16 (Epistemic status of an answer (Huang et al. 2005)) *Given an ontology Σ and a query ϕ, the answer to ϕ will have one of the four epistemic states:*

1. *Over-determined:* $\Sigma \mid\!\approx \phi$ *and* $\Sigma \mid\!\approx \neg\phi$;
2. *Accepted:* $\Sigma \mid\!\approx \phi$ *and* $\Sigma \not\mid\!\approx \neg\phi$;
3. *Rejected:* $\Sigma \not\mid\!\approx \phi$ *and* $\Sigma \mid\!\approx \neg\phi$;
4. *Undetermined:* $\Sigma \not\mid\!\approx \phi$ *and* $\Sigma \not\mid\!\approx \neg\phi$.

If we regard the relation $\mid\!\approx$ as "justified membership" of instances to concepts (see Def. 11), $\Sigma \mid\!\approx \phi$ corresponds to a YES answer to query ϕ w.r.t. program $\mathcal{T}(\Sigma)$.

Property 5 *Let $\mid\!\approx$ be the "justified membership" of instances to concepts relationship. Let Σ be a δ-ontology. The answer to a query ϕ is never over-determined.*

Proof: Suppose by the contrary that the answer to ϕ is over-determined. Then it must be the case that there exist two warranted arguments $\langle \mathcal{A}, \phi \rangle$ and $\langle \mathcal{A}, \sim\phi \rangle$ w.r.t. $\mathcal{T}(\Sigma)$. This cannot be the case in DeLP as shown by García & Simari (2004).

Notice that, as required by traditional DL reasoning, DeLP does not adopt the closed-world assumption. That is, not being able to prove Q in DeLP does not imply that $\sim Q$ will be assumed. On the contrary, such an answer will be the result of a dialectical analysis that will take into account all of the reasons for Q and $\sim Q$.

Definition 17 (Soundness (Huang et al. 2005)) *An inconsistency reasoner $\mid\!\approx$ is* sound *if the formulas that follow from an inconsistent theory Σ follow from a consistent subtheory of Σ using classical reasoning.*

Property 6 $\mid\!\approx$ *is a sound inconsistency reasoner.*

Proof: Let $\Sigma = (T_S, T_D, A)$ be a δ-ontology. If $\Sigma \mid\!\approx \phi$ then there exists a warranted argument $\langle \mathcal{A}, \phi \rangle$ w.r.t. $\mathcal{T}(\Sigma) = (\Pi_S \cup \Pi_A, \Delta)$, where $\Pi_S = \mathcal{T}_\Pi(T_S)$, $\Pi_A = \mathcal{T}_\Pi(A)$ and $\Delta = \mathcal{T}_\Delta(T_D)$. The set $\Pi_S \cup \Pi_A \cup \mathcal{A}$ (see Def. 2) is consistent and as $\mathcal{T}$ is a transformation that preserves semantics (Grosof et al. 2003), there must exists a subset $\Sigma' \subseteq \Sigma$ such that $\mathcal{T}(\Sigma') = \Pi_S \cup \Pi_A \cup \mathcal{A}$.

Definition 18 (Consistency (Huang et al. 2005)) *An inconsistency reasoner $\mid\!\approx$ is* consistent *iff $\Sigma \mid\!\approx \phi \Rightarrow \Sigma \not\mid\!\approx \neg\phi$.*

Property 7 $\mid\!\approx$ *is a consistent inconsistency reasoner.*

Proof: Corollary of Prop. 5.

Definition 19 (Meaningfulness (Huang et al. 2005)) *An answer given by an inconsistency reasoner is meaningful iff it is consistent and sound. An inconsistency reasoner is said to be meaningful iff all of its answers are meaningful.*

Property 8 $\mid\!\approx$ *is a meaningful inconsistency reasoner.*

Proof: Trivial from Props. 6 and 7.

Implementation issues

As mentioned in Section 4, we base our translation function from DL to DeLP on the work reported by Grosof et al. (2003). In this respect, Volz (2004) shows that the fragment of DL expressible in logic programming (referred to as *DLP*) is sufficient to express most available Web ontologies. Volz has analyzed the largest currently available collection of Web ontologies and checked which fragment of those ontologies can be expressed in DLP; he claims both that DLP languages suffice to express 77%–87% of the analyzed ontologies and can express 93%–99% of the individual axioms in the analyzed ontologies.

As performing defeasible argumentation is a computationally complex task, an abstract machine called JAM (Justification Abstract Machine) has been specially developed for an efficient implementation of DeLP (García & Simari 2004). JAM provides an argument-based extension of the traditional WAM (Warren's Abstract Machine) for Prolog. A full-fledged implementation of DeLP is available online[3], including facilities for visualizing arguments and dialectical trees.

8 Related Work

Grosof et al. (2003) show how to interoperate, semantically and inferentially, between the leading Semantic Web approaches to rules (RuleML Logic Programs) and ontologies (OWL DL) by analyzing their expressive intersection. They define a new intermediate knowledge representation called Description Logic Programs (DLP), and the closely related Description Horn Logic (DHL) which is an expressive fragment of FOL. They show how to perform the translation of premises and inferences from the DLP fragment of DL to logic programming. Part of our approach is based on Grosof's work as the algorithm for translating DL ontologies into DeLP is based on it. However, as Grosof et al. (2003) use standard Prolog rules, they are not able to deal with inconsistent DL knowledge bases as our proposal does.

Heymans & Vermeir (2002) extend the DL $\mathcal{SHOQ}(D)$ 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, introducing nonmonotonicity into $\mathcal{SHOQ}(D)$. Similarly to Heymans & Vermeir (2002) we allow to perform inferences inferences from inconsistent ontologies by considering subsets (arguments) of the original ontology. Heymans & Vermeir (2002) also impose a hard-coded comparison criterion on DL axioms. In our work, the system, and not the programmer, decides which DL axioms are to be preferred as we use specificity as argument comparison criterion. We think that our approach can be considered more declarative in this respect. In particular the comparison criterion in DeLP is modular, so that rule comparison could also be adopted (García & Simari 2004).

Eiter et al. (2004) propose a combination of logic programming under the answer set semantics with the DLs $\mathcal{SHIF}(D)$ and $\mathcal{SHOIN}(D)$. This combination allows for building rules on top of ontologies. In contrast to our approach, they keep separated rules and ontologies and handle exceptions by codifying them explicitly in programs under answer set semantics.

Huang et al. (2005) use paraconsistent logics to reason with inconsistent ontologies. They use a selection function to determine which consistent subsets of an inconsistent ontology should be considered in the reasoning process. In our approach given an inconsistent ontology Σ, we consider the set of warranted arguments from $\mathcal{T}(\Sigma)$ as the valid consequences.

Williams & Hunter (2007) use argumentation to reason with possibly inconsistent rules on top of DL ontologies. In contrast, we translate possible inconsistent DL ontologies to DeLP to reason with them within DeLP. Laera et al. (2006) propose an approach for supporting the creation and exchange of different arguments, that support or reject possible correspondences between ontologies in the context of a

[3]See `http://lidia.cs.uns.edu.ar/DeLP`

multi-agent system. In our work we assume correspondences between ontologies as given.

Antoniou & Bikakis (2007) propose a rule-based approach to defeasible reasoning based on a translation to logic programming with declarative semantics that can reason with rules, RDF(S) and parts of OWL ontologies. In contrast with our approach, argumentation is not used explicitly for detecting inconsistent ontologies. Instead, they translate OWL statements to Prolog to reason with Defeasible Logic.

9 Conclusions and Future Work

We have presented a framework for reasoning with inconsistent Description Logics (DL) ontologies. Our proposal involves expressing DL ontologies as a Defeasible Logic Program (DeLP) by means of a translation function $\mathcal{T}$. Given a query ϕ posed w.r.t. an inconsistent ontology Σ, a dialectical analysis is performed on a DeLP program $\mathcal{T}(\Sigma)$ where all arguments in favor and against ϕ's acceptance are taken into account. We have also presented an application to ontology integration based on the global-as-view approach to ontology integration where queries respect a global ontology are posed while data is extracted from local ontologies that could be inconsistent.

Several issues need to be solved and part of our efforts are focused on that matter. For instance, as DeLP does not support disjunctions in the head of rules, our approach is not able to deal with DL axioms that require the construction of such rules, a possible extension to this work could be in that direction. Other issue that needs to be addressed is given by the mapping of DL equality 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 argument construction when solving queries in actual DeLP programs. Nevertheless the examples considered in this work model an important part of ontologies where this situation does not happen, a possible solution involves separating equality axioms from simple inclusion axioms and keeping track of their instantiations into ground rules to avoid such looping situations.

References

Antoniou, G. & Bikakis, A. (2007), 'DR-Prolog: A System for Defeasible Reasoning with Rules and Ontologies on the Semantic Web', *IEEE Trans. on Knowledge and Data Eng.* **19**(2), 233–245.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D. & Patel-Schneider, P., eds (2003), *The Description Logic Handbook – Theory, Implementation and Applications*, Cambridge University Press.

Berners-Lee, T., Hendler, J. & Lassila, O. (2001), 'The Semantic Web', *Scientific American* **284**(5), 34–43.

Brewka, G., Dix, J. & Konolige, K. (1997), *Non monotonic reasoning. An overview*, CSLI Publications, Stanford, USA.

Calvanese, D., Giacomo, G. D. & Lenzerini, M. (2001), A Framework for Ontology Integration, *in* 'Proceedings of the 1st Semantic Web Working Symposium (SWWS 2001)', pp. 303–316.

Caminada, M. (2008), 'On the Issue of Contraposition of Defeasible Rules', *COMMA 2008 (to appear)* .

Cecchi, L. A., Fillottrani, P. R. & Simari, G. R. (2006), On Complexity of DeLP through Game Semantics, *in* J. Dix & A. Hunter, eds, '11th. Intl. Workshop on Nonmonotonic Reasoning', pp. 386–394.

Chesñevar, C. I., Maguitman, A. & Loui, R. (2000), 'Logical Models of Argument', *ACM Computing Surveys* **32**(4), 337–383.

Eiter, T., Lukasiewicz, T., Schindlauer, R. & Tompits, H. (2004), 'Combining Answer Set Programming with Description Logics for the Semantic Web', *KR 2004* pp. 141–151.

García, A. J. & Simari, G. R. (2004), 'Defeasible Logic Programming: An Argumentative Approach', *Theory and Practice of Logic Programming* **4**(1), 95–138.

Grosof, B. N., Horrocks, I., Volz, R. & Decker, S. (2003), 'Description Logic Programs: Combining Logic Programs with Description Logics', *WWW2003, Budapest, Hungary* .

Gruber, T. R. (1993), 'A translation approach to portable ontologies', *Knowledge Acquisition* **5**(2), 199–220.

Haarslev, V. & Möller, R. (2001), RACER System Description, Technical report, University of Hamburg, Computer Science Department.

Heymans, S. & Vermeir, D. (2002), A Defeasible Ontology Language, *in* 'CoopIS/DOA/ODBASE', pp. 1033–1046.

Huang, Z., van Harmelen, F. & ten Teije, A. (2005), Reasoning with inconsistent ontologies, *in* 'Proc. of the Nineteenth Intl. Joint Conference on Artificial Intelligence (IJCAI'05)', pp. 454–459.

Klein, M. (2001), Combining and relating ontologies: an analysis of problems and solutions, *in* A. Gomez-Perez, M. Gruninger, H. Stuckenschmidt & M. Uschold, eds, 'Workshop on Ontologies and Information Sharing, IJCAI'01', Seattle, USA.

Laera, L., Tamma, V., Euzenat, J., Bench-Capon, T. & Payne, T. (2006), Reaching agreement over ontology alignments, *in* 'Proceedings of the 5th International Semantic Web Conference (ISWC 2006), Athens, GA'.

McGuiness, D. L. & van Harmelen, F. (2004), 'OWL Web Ontology Language Overview'.

Mitra, P. (2004), An Algebraic Framework for the Interoperation of Ontologies, PhD thesis, Dept. of Electrical Eng., Stanford Univ.

Prakken, H. & Vreeswijk, G. (2002), Logics for Defeasible Argumentation, *in* D. Gabbay & F. Guenthner, eds, 'Handbook of Philosophical Logic', Kluwer Academic Publisher, pp. 219–318.

Simari, G. R. & Loui, R. P. (1992), 'A Mathematical Treatment of Defeasible Reasoning and its Implementation', *Artificial Intelligence* **53**, 125–157.

Volz, R. (2004), Web Ontology Reasoning with Logic Databases, PhD thesis, Universität Fridericiana zu Karlsruhe.

Williams, M. & Hunter, A. (2007), 'Harnessing ontologies for argument-based decision-making in breast cancer', *Proc. of the Intl. Conf. on Tools with AI (ICTAI'07)* pp. 254–261.