Applied Artificial Intelligence
Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t713191765

REASONING WITH INCONSISTENT ONTOLOGIES THROUGH ARGUMENTATION

Sergio Alejandro Gómez [a]; Carlos Iván Chesñevar [ab]; Guillermo Ricardo Simari [a]
[a] Artificial Intelligence Research and Development Laboratory, Department of Computer Science and Engineering, Universidad Nacional del Sur, Bahía Blanca, Argentina [b] CONICET (National Council of Scientific and Technical Research), Argentina

Online publication date: 29 January 2010

To cite this Article Alejandro Gómez, Sergio, Iván Chesñevar, Carlos and Simari, Guillermo Ricardo(2010) 'REASONING WITH INCONSISTENT ONTOLOGIES THROUGH ARGUMENTATION', Applied Artificial Intelligence, 24: 1, 102 — 148
To link to this Article: DOI: 10.1080/08839510903448692
URL: http://dx.doi.org/10.1080/08839510903448692

Applied Artificial Intelligence, 24:102–148

ISSN: 0883-9514 print/1087-6545 online
DOI: 10.1080/08839510903448692

Taylor & Francis
Taylor & Francis Group

REASONING WITH INCONSISTENT ONTOLOGIES THROUGH ARGUMENTATION

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

[1]*Artificial Intelligence Research and Development Laboratory, Department of Computer Science and Engineering, Universidad Nacional del Sur, Bahía Blanca, Argentina*

[2]*CONICET (National Council of Scientific and Technical Research), Argentina*

□ *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 article, we present δ-ontologies, 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 an 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.*

INTRODUCTION AND MOTIVATIONS

The semantic web (Berners-Lee, Hendler, and Lassila 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. In the context of knowledge-sharing, the term ontology means a specification of a conceptualization. That is, an ontology is a description of the concepts and relationships that can exist for an agent or a community of agents (Gruber 1993).

This research was funded by TIN2006-15662-C02-01 (MEC, Spain), PGI 24/ZN10 (SGCyT, UNS, Argentina), by CONICET (Project PIP 112-200801-02798), and by COST Action IC0801 on Agreement Technologies (ESF – European Union). The present article unifies and further develops the content of Gómez et al. (2006, 2008a).

Address correspondence to Sergio Alejandro Gómez, Departamento de Ciencias e Ingenieria de la Computacion, Universidad Nacional del Sur, Av. Alem 1253, Bahía Blanca 8000, Argentina. E-mail: sag@cs.uns.edu.ar

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 and van Harmelen 2004), whose semantics are based on Description Logics (DL) (Baader et al. 2003). Description logic ontologies are comprised of a terminology composed of a set of axioms defining a set of classes as well as a set of assertions about which individuals are known to be members of those classes and establishing relations/properties between individuals.

As pointed out by Wang et al. (2005), one of the advantages of logic-based ontology languages, is that reasoners can be used to compute subsumption relationships between classes and to identify unsatisfiable (inconsistent) classes. With the maturation of the tableaux algorithm based DL reasoners (such as Racer (Haarslev and Möller 2001), FaCT (Horrocks 1998), Pellet (Parsia and Sirin 2004)), it is possible to perform efficient reasoning on large ontologies formulated in expressive DL. When checking satisfiability (consistency) most modern DL reasoners can only provide lists of unsatisfiable classes and offer no further explanation for their unsatisfiability. (A notable exception to is Horridge, Parsia, and Sattler (2008) discussed later in this article).

There are two main (but not incompatible) ways to deal with inconsistency in ontologies (Huang, van Harmelen, and ten Teije 2005): one is to diagnose and repair it when it is encountered; another is to avoid the inconsistency and to apply a nonstandard inference relation to obtain meaningful answers. Although there are existing approaches for the former (e.g., identifying the minimally unsatisfiable subontologies or calculating the maximally satisfiable subontologies (Lam, Pan, Sleeman, and Vasconcelos 2006)), performing it can be very difficult. That is, the process of "debugging" an ontology (i.e., determining why classes are unsatisfiable) is left for the user. However, when faced with several unsatisfiable classes in a moderately large ontology, even expert ontology engineers can find it difficult to work out the underlying error (Wang et al. 2005). In this work we will focus on the latter approach.

We propose to use defeasible argumentation (Chesñevar, Maguitman, and Loui 2000; Prakken and Vreeswijk 2002; Bench-Capon and Dunne 2007) to reason with inconsistent ontologies. In particular, *DeLP* is an argumentative framework based on logic programming which is capable of dealing with possibly inconsistent knowledge bases (KB) codified as a set of horn-like clauses called DeLP programs (García and 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 called *δ-ontologies* for representing and reasoning with possibly inconsistent DL ontologies. For this we interpret

DL ontologies as DeLP programs adapting (Grosof, Horrocks, Volz, and Decker 2003), which shows how a subset of DL can be effectively translated into an equivalent subset of Horn logic.

As already mentioned, traditional DL ontologies (in the sense of Baader et al. (2003)) are comprised of a terminology composed of a set of axioms defining a set of classes and a set of assertions about individuals in those classes. Thus, δ-ontologies are DL ontologies where the terminology is partitioned in such a way that conflicting axioms involved with the unsatisfiability of classes are treated specially as defeasible ones. Reasoning with such ontologies will then be carried out by means of a dialectical analysis. Our proposal involves interpreting DL ontologies as DeLP programs. That is, given a DL ontology Σ_{DL}, provided that it satisfies certain restrictions, it will be translated into a DeLP program Σ_{DeLP}. Given a query ϕ posed w.r.t. Σ_{DL} about the membership of an individual a to a concept C, a dialectical process will be performed to determine if the literal $C(a)$ is warranted w.r.t. Σ_{DeLP}.

We also present an application of our framework to the problem of ontology integration. Reuse of existing ontologies is often not possible without considerable effort. When one wants to reuse different ontologies together, those ontologies have to be combined in some way. This can be done by integrating the ontologies, which means that they are merged into one new ontology, or the ontologies can be kept separate. In both cases, the ontologies have to be aligned, which means that they have to be brought into mutual agreement (Klein 2001). A particular source of inconsistency is related to the use of imported ontologies when the knowledge engineer has no authority to correct them, and as these imported ontologies are usually developed independently, their combination could also result in inconsistencies. One kind of such integration is known as global-as-view integration (Calvanese, Giacomo, and Lenzerini 2001), where a global ontology is used as a view of local ontologies that define the actual data. We apply our proposal to perform global-as-view integration when the involved ontologies can be potentially inconsistent.

DESCRIPTION LOGICS

Description logics are a well-known family of knowledge representation formalisms (Baader et al. 2003). They are based on the notions of concepts (unary predicates representing classes, noted as Bird, Fly, etc.) and roles (binary relations noted as eats, isParentOf, etc.), 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 $\mathcal{SHIF}$ 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 ($\forall R.C$). Besides, the expressions $\top$ and $\bot$ are shorthand 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. Another extension involves the possibility of allowing individual names (or nominals) in the description language; the most basic is the "set" (or one-of) constructor, written $\{a_1, \ldots, a_n\}$ where $a_1, \ldots, a_n$ are individual names.

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 (e.g., $\mathsf{Bird} \sqsubseteq \mathsf{Fly}$). 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 $a : C$ (e.g., $\mathsf{OPUS} : \mathsf{Bird}$) 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 (e.g., $\langle \mathsf{ADAM}, \mathsf{ABEL} \rangle : \mathsf{isParentOf}$).

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 $I = \langle \Delta^I, \cdot^I \rangle$ consists of a nonempty 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 R to a subset R^I of $\Delta^I \times \Delta^I$. The interpretation function is extended to arbitrary concept descriptions as follows: $(\neg C)^I = \Delta^I \backslash C^I$; $(C \sqcup D)^I = C^I \cup D^I$; $(C \sqcap D)^I = C^I \cap D^I$; $(\exists R.C)^I = \{x \mid \exists y \text{ s.t. } (x, y) \in R^I \text{ and } y \in C^I\}$, and $(\forall R.C)^I = \{x \mid \forall y, (x, y) \in R^I \text{ implies } y \in C^I\}$.

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 $C^I = D^I$. 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.

A knowledge representation system based on DL is able to perform specific kinds of reasoning, its purpose goes beyond storing concept definitions and assertions (the architecture of such a system is presented in Figure 1). As a DL ontology has semantics that makes it equivalent to

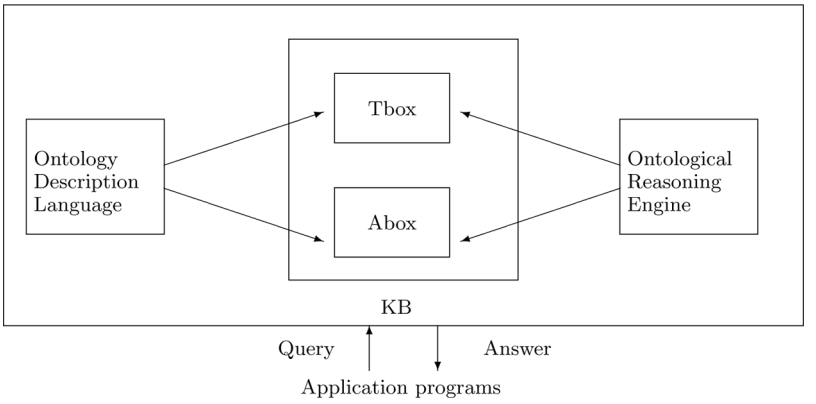

FIGURE 1 Architecture of a knowledge representation system based on Description Logics (adapted from Baader et al. (2003), Fig. 2.1).

a set of axioms of first-order logic, it contains implicit knowledge that can be made explicit through inferences. Inferences in DL systems are usually divided into Tbox reasoning and Abox reasoning. In this article we are concerned only with Abox reasoning, so we refer the interested reader to Baader et al. (2003).

- *Consistency:* An Abox A is consistent w.r.t. Tbox T if there exists an interpretation which is a model of both A and T.
- *Instance checking:* Instance checking consists of determining if an assertion is entailed from an Abox. For instance, $T \cup A \models a : C$ indicates that the individual a is a member of the concept C w.r.t. the Abox A and the Tbox T.
- *Retrieval:* If we consider a knowledge base as a means to store information about individuals, we might want to know all the individuals that are instances of a particular concept description. Given an ontology (T, A) and a concept description C, in the retrieval problem we are interested in knowing all the individuals a such that $T \cup A \models a : C$.

We now define the notion of inconsistency in DL ontologies.

Definition 2.1 (Inconsistency in DL ontologies (Baader et al. 2003))**.** An ontology $\Sigma = (T, A)$ is inconsistent iff for every interpretation $I = \langle \Delta^I, \cdot^I \rangle$ of Σ, Δ^I is empty.

Huang et al. (2004, 2005) survey several scenarios that may cause inconsistency, such as mis-presentation of defaults, polysemy (i.e., the capacity for a sign to have different meanings), ontology migration from another formalism, and use of multiple sources. We now present

some inconsistent ontologies that will serve as motivating examples; they correspond to typical examples from the literature of nonmonotonic reasoning in Artificial Intelligence, and will be analyzed from an argumentative perspective in the next Section. Notice that following DL usual conventions, concepts are noted beginning with capital letters (e.g., Bird, Fly, etc.), roles with lowercase letters (e.g., in_fusion, etc.), and individual names with uppercase letters (e.g., OPUS, ACME, STEEL, etc.).

Example 2.1. Let $\Sigma_{2.1} = (T, A)$ be an ontology, where

$$T = \left\{ \begin{array}{l} \mathsf{Penguin} \sqsubseteq \mathsf{Bird} \\ \mathsf{Bird} \sqsubseteq \mathsf{Fly} \\ \mathsf{Bird} \sqcap \mathsf{Broken\_Wing} \sqsubseteq \neg\mathsf{Fly} \\ \mathsf{Super\_Penguin} \sqsubseteq \mathsf{Penguin} \sqcap \mathsf{Fly} \end{array} \right\},$$

and

$$A = \left\{ \begin{array}{l} \mathsf{OPUS} : \mathsf{Super\_Penguin} \\ \mathsf{OPUS} : \mathsf{Broken\_Wing}. \end{array} \right\}.$$

The Tbox T says that penguins are birds; birds can fly unless they have a broken wing, and superpenguins are penguins capable of flying. The Abox A asserts that it is known that Opus is a superpenguin having a broken wing.

Example 2.2. Let us consider the Nixon's diamond problem, a well-known problem in nonmonotonic reasoning (Reiter and Criscuolo 1981). Let $\Sigma_{2.2} = (T, A)$ be a DL ontology where[2]

$$T = \left\{ \begin{array}{l} \mathsf{Lives\_In\_Chicago} \sqsubseteq \mathsf{Has\_A\_Gun} \\ \mathsf{Lives\_In\_Chicago} \sqcap \mathsf{Pacifist} \sqsubseteq \neg\mathsf{Has\_A\_Gun} \\ \mathsf{Quaker} \sqsubseteq \mathsf{Pacifist} \\ \mathsf{Republican} \sqsubseteq \neg\mathsf{Pacifist} \end{array} \right\},$$

and

$$A = \left\{ \begin{array}{l} \mathsf{NIXON} : \mathsf{Lives\_In\_Chicago} \\ \mathsf{NIXON} : \mathsf{Quaker} \\ \mathsf{NIXON} : \mathsf{Republican} \end{array} \right\}.$$

The meaning of this ontology is as follows. If someone lives in Chicago, then he has a gun unless he is a pacifist; Quakers are pacifists, and Republicans are not pacifists. Nixon lives in Chicago and he is both a Quaker and a Republican.

Example 2.3. Let us consider the $\Sigma_{2.3} = (T, A)$ ontology about the stock market domain:

$$T = \left\{ \begin{array}{l} \mathsf{Good\_Price} \sqsubseteq \mathsf{Buy\_Stock} \\ \mathsf{Good\_Price} \sqcap \mathsf{Risky\_Company} \sqsubseteq \neg \mathsf{Buy\_Stock} \\ \exists \mathsf{in\_fusion}.\top \sqcup \mathsf{Closing} \sqsubseteq \mathsf{Risky\_Company} \\ \exists \mathsf{in\_fusion}.\mathsf{Strong} \sqsubseteq \neg \mathsf{Risky\_Company} \end{array} \right\},$$

and

$$A = \left\{ \begin{array}{l} \mathsf{ACME} : \mathsf{Good\_Price} \\ \langle \mathsf{ACME}, \mathsf{STEEL} \rangle : \mathsf{in\_fusion} \\ \mathsf{STEEL} : \mathsf{Strong} \end{array} \right\}.$$

The meaning of this ontology is as described next. If some company's stock has a good price, then an investor should buy it unless that company is risky. Risky companies are those which are in the process of fusion or closing down, except for those that are in fusion with a strong company. It is known that Acme's stocks have a good price, and that Acme is in fusion with a strong company named Steel.

Example 2.4. Let $\Sigma_{2.4} = (T, A)$ be an ontology about the Royal African elephants (Sandewall 1986), where:

$$T = \left\{ \begin{array}{l} \mathsf{Elephant} \sqsubseteq \mathsf{Gray} \\ \mathsf{Royal\_Elephant} \sqsubseteq \neg \mathsf{Gray} \\ \mathsf{Royal\_Elephant} \sqcup \mathsf{African\_Elephant} \sqsubseteq \mathsf{Elephant} \end{array} \right\},$$

and

$$A = \left\{ \begin{array}{l} \mathsf{CLYDE} : \mathsf{Royal\_Elephant} \\ \mathsf{CLYDE} : \mathsf{African\_Elephant} \end{array} \right\}.$$

It says that elephants are gray but royal elephants are not. Royal elephants are elephants; African elephants are elephants as well. It is known that Clyde is both a Royal and an African elephant.

Example 2.5. Let $\Sigma_{2.5} = (T, A)$ be an ontology about the adult students problem (Geffner and Pearl 1990), with

$$T = \left\{ \begin{array}{l} \mathsf{Adult} \sqsubseteq \mathsf{Worker} \\ \mathsf{University\_Student} \sqsubseteq \neg \mathsf{Worker} \\ \mathsf{University\_Student} \sqsubseteq \mathsf{Adult} \end{array} \right\},$$

and

$$A = \left\{ \begin{array}{l} \mathsf{KEN : Adult} \\ \mathsf{KEN : University\_Student} \end{array} \right\}.$$

It expresses that adult people are workers, and that university students do not work and are adults. It also asserts that Ken is both an adult and a university student.

Example 2.6 (adapted from Huang et al. (2004))**.** Let us consider the ontology $\Sigma_{2.6} = (T, A)$ adapted from Huang et al. (2004, Section 2.2) as an example of inconsistency caused by polysemy, with:

$$T = \left\{ \begin{array}{l} \mathsf{Married\_Woman} \sqsubseteq \mathsf{Woman} \\ \mathsf{Married\_Woman} \sqsubseteq \neg \mathsf{Divorcee} \\ \mathsf{Divorcee} \sqsubseteq \mathsf{Had\_Husband} \sqcap \neg \mathsf{Has\_Husband} \\ \mathsf{Has\_Husband} \sqsubseteq \mathsf{Married\_Woman} \\ \mathsf{Had\_Husband} \sqsubseteq \mathsf{Married\_Woman} \\ \mathsf{Had\_Husband} \sqsubseteq \neg \mathsf{Has\_Husband} \end{array} \right\},$$

and

$$A = \left\{ \begin{array}{l} \mathsf{FLOR : Married\_Woman} \\ \mathsf{LETICIA : Divorcee} \end{array} \right\}.$$

This ontology expresses that a married woman is a woman, a married woman is not a divorcee, a divorcee had a husband and has no husband. "Has_Husband" means married, "Had_Husband" also means married, and that somebody had a husband implies she does not have a husband. It is also known that Flor is a married woman and Leticia is divorced.

Notice that the concept "Divorcee" is unsatisfiable, because of the misuse of the word "Married_Woman." Therefore, one has to carefully check if there is some misunderstanding with respect to concepts that have been used in the ontology; when an ontology is large, this kind of requirement may become rather difficult (Huang et al. 2004, p. 7).

In the next section, we will see how the above-described situations can be handled in DeLP.

DEFEASIBLE LOGIC PROGRAMMING

When a rule supporting a conclusion may be defeated by new information, it is said that such reasoning is defeasible (Pollock 1974, 1987; Nute 1988; Pollock 1995; Simari and Loui 1992). When defeasible reasons

or rules are chained to reach a conclusion, we have arguments instead of proofs. Arguments may compete, rebutting each other, so that a process of argumentation is a natural result of the search for arguments. Adjudication of competing arguments must be performed, comparing arguments in order to determine what beliefs are ultimately accepted as warranted or justified. Preference among conflicting arguments is defined in terms of a preference criterion which establishes a relation "$\preceq$" among possible arguments; thus, for two arguments $\mathcal{A}$ and $\mathcal{B}$ in conflict, it may be the case that $\mathcal{A}$ is strictly preferred over $\mathcal{B}$ ($\mathcal{A} \succ \mathcal{B}$), that $\mathcal{A}$ and $\mathcal{B}$ are equally preferable ($\mathcal{A} \succeq \mathcal{B}$ and $\mathcal{A} \preceq \mathcal{B}$), or that $\mathcal{A}$ and $\mathcal{B}$ are not comparable with each other. In the above setting, since we arrive at conclusions by building defeasible arguments, and since logical argumentation is usually referred to as argumentation, we sometimes call this kind of reasoning defeasible argumentation.

The growing success of argumentation-based approaches has caused a rich cross-breeding with other disciplines, providing interesting results in different areas such as group decision-making (Zhang, Sun, and Chen 2005), knowledge engineering (Carbogim, Robertson, and Lee 2000), legal reasoning (Prakken and Sartor 2002; Verheij 2005), and multiagent systems (Parsons, Sierrra, and Jennings 1998; Sierra and Noriega 2002; Rahwan et al. 2003), among others. During the last decade several defeasible argumentation frameworks have been developed, most of them on the basis of suitable extensions to logic programming (see Chesñevar et al. (2000), Prakken and Vreeswijk (2002), and Kakas and Toni (1999)). Defeasible logic programming (García and Simari 2004) is one of such formalisms, combining results from defeasible argumentation theory (Simari and Loui 1992) and logic programming (Lloyd 1987). DeLP is a suitable framework for building real-world applications that have proven to be particularly attractive in that context, such as clustering (Gómez and Chesñevar 2004), intelligent web search (Chesñevar and Maguitman 2004b; Chesñevar, Maguitman, and Simari 2006), knowledge management (Chesñevar et al. 2005a,b), multiagent systems (Brena, Chesñevar and Aguirre 2006), natural language processing (Chesñevar and Maguitman 2004a), intelligent web forms (Gómez, Chesñevar, and Simari 2008b), among others.

Defeasible logic programming (García and Simari 2004) provides a language for knowledge representation and reasoning that uses defeasible argumentation (Chesñevar et al. 2000; Prakken and Vreeswijk 2002; Simari and Loui 1992) to decide between contradictory conclusions through a dialectical analysis. The architecture of a knowledge representation system based on defeasible argumentation is shown in Figure 2 (the analogies of this architecture w.r.t. the architecture presented in Figure 1 will be considered later.

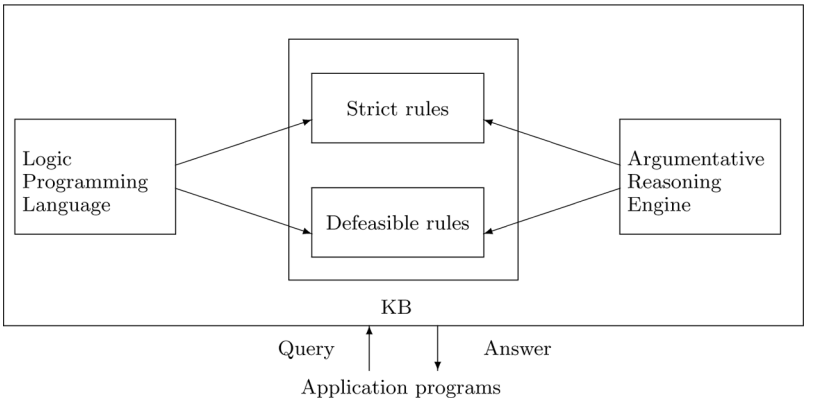

FIGURE 2 Architecture of a knowledge representation system based on defeasible argumentation.

In a defeasible logic program $\mathcal{P} = (\Pi, \Delta)$, a set Π of strict rules $P \leftarrow Q_1, \ldots, Q_n$, and a set Δ of defeasible rules $P \prec Q_1, \ldots, Q_n$ can be distinguished.

Definition 3.1 (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 $L = p$. Notice that there is an extension to DeLP that allows to define presumptions (or defeasible rules without body) that model defeasible facts (see García and Simari (2004, Sect. 6.2)); however, they are outside the scope of this work.

Deriving literals in DeLP results in the construction of arguments. An argument $\mathcal{A}$ is a (possibly empty) set of ground (i.e., without variables) defeasible rules that together with the set Π provides a logical proof for a given literal Q, satisfying the additional requirements of noncontradiction and minimality. Formally:

Definition 3.2 (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 noncontradictory (i.e., $\Pi \cup \mathcal{A}$ does not entail two complementary literals P and $\sim P$); and (3) there is no $\mathcal{A}' \subset \mathcal{A}$ 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 *subargument* 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" on argument construction. The noncontradiction 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 3.3 (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 and Loui 1992) is typically used as a syntax-based preference criterion among conflicting arguments, favoring those arguments which are more informed or more direct (Simari and Loui 1992; Stolzenburg, García, Chesñevar, and Simari 2003). For example, let us consider three arguments $\langle \{a \prec b, c\}, a \rangle$, $\langle \{\sim a \prec b\}, \sim a \rangle$, and $\langle \{(a \prec b); (b \prec c)\}, a \rangle$ built on the basis of a program $\mathcal{P} = (\Pi, \Delta) = (\{b, c\}, \{b \prec c; a \prec b; a \prec b, c; \sim a \prec b\})$. When using generalized specificity as the comparison criterion between arguments, the argument $\langle \{a \prec b, c\}, a \rangle$ would be preferred over the argument $\langle \{\sim a \prec b\}, \sim a \rangle$ as the former is considered more informed (i.e., it relies on more premises). However, argument $\langle \{\sim a \prec b\}, \sim a \rangle$ is preferred over $\langle \{(a \prec b); (b \prec c)\}, a \rangle$ as the former is regarded as more direct (i.e., it is obtained from a shorter derivation). However, it must be remarked that besides specificity, other alternative preference criteria could also be used; e.g., considering numerical values corresponding to necessity measures attached to argument conclusions (Chesñevar, Simari, Alsinet, and Godo 2004; Chesñevar, Simari, Godo, and Alsinet 2005) or defining argument comparison using rule priorities. This last approach is used in D-Prolog (Nute 1988), defeasible logic (Nute 1992), extensions of defeasible logic (Antoniou et al. 2000, 1998), and logic programming without negation as failure (Kakas, Mancarella, and Dung 1994; Dimopoulos and Kakas 1995).

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 $[\langle \mathcal{A}_0, Q_0 \rangle, \langle \mathcal{A}_1, Q_1 \rangle, \langle \mathcal{A}_2, Q_2 \rangle, \ldots, \langle \mathcal{A}_n, Q_n \rangle \ldots]$ 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 $\mathcal{P}$.

We present a property that will be useful for proving some results about our proposal of reasoning with δ-ontologies.

Proposition 3.1 (García and Simari 2004). *For any program $\mathcal{P}$ in DeLP, it cannot be the case that both Q and $\sim Q$ are warranted.*

In the next sections we will present a method for expressing arbitrary DL ontologies in DeLP. For now on, based on the fact discovered by Grosof et al. (2003) that DL inclusion axioms of the form "$\mathsf{C} \sqsubseteq \mathsf{D}$" can be equated to PROLOG rules of the form "D(X) :- C(X)", we will show how the application problems modeled by the ontologies presented in the previous section can be expressed in DeLP in such a way that it could be possible for an agent to automatically reason in such cases. Notice also that

following PROLOG usual conventions, predicates are noted with lowercase letters (e.g., *bird*, *fly*, etc.), variable names with capital letters (e.g., X, Y, Z, etc.), and constant names with lowercase letters (e.g., *opus*, *acme*, *steel*, etc.).

Example 3.1. Here follows the DeLP program $\mathcal{P}_{3.1} = (\Pi, \Delta)$ which models the situation presented in Example 2.1:

$$\Pi = \left\{ \begin{array}{l} bird(X) \leftarrow penguin(X) \\ penguin(X) \leftarrow super\_penguin(X) \\ super\_penguin(opus) \\ broken\_wing(opus) \end{array} \right\},$$

and

$$\Delta = \left\{ \begin{array}{l} \sim fly(X) \prec bird(X), broken\_wing(X) \\ fly(X) \prec bird(X) \\ fly(X) \prec super\_penguin(X) \end{array} \right\}.$$

In DeLP, the answer for $fly(opus)$ is *Yes*, while the answer for $\sim fly(opus)$ is *No* as it is shown next. It is possible to build an argument $\langle \mathcal{A}_1, fly(opus) \rangle$, where

$$\mathcal{A}_1 = \{fly(opus) \prec bird(opus)\}.$$

This argument is properly defeated by another argument $\langle \mathcal{A}_2, \sim fly(opus) \rangle$, with

$$\mathcal{A}_2 = \{\sim fly(opus) \prec bird(opus), broken\_wing(opus)\}.$$

However, argument $\mathcal{A}_1$ is reinstated by another argument $\langle \mathcal{A}_3, fly(opus) \rangle$ which is a blocking defeater for $\mathcal{A}_2$, where

$$\mathcal{A}_3 = \{fly(opus) \prec super\_penguin(opus)\}.$$

Example 3.2 (taken from García and Simari (2004))**.** Let us present the DeLP program $\mathcal{P}_{3.2} = (\Pi, \Delta)$ which represents the ontology presented in Example 2.2:

$$\Pi = \left\{ \begin{array}{l} lives\_in\_chicago(nixon) \\ quaker(nixon) \\ republican(nixon), \end{array} \right\},$$

and

$$\Delta = \left\{ \begin{array}{l} has\_a\_gun(X) \prec lives\_in\_chicago(X) \\ \sim has\_a\_gun(X) \prec lives\_in\_chicago(X), pacifist(X) \\ pacifist(X) \prec quaker(X) \\ \sim pacifist(X) \prec republican(X) \end{array} \right\}.$$

For this particular program, García and Simari (2004, Example 2.2) show that the answer for queries $pacifist(nixon)$ and $\sim pacifist(nixon)$ is *Undecided*.

Example 3.3 (taken from García and Simari (2004))**.** Consider the DeLP program $\mathcal{P}_{3.3} = (\Pi, \Delta)$ which represents the situation posed by Example 2.3, where

$$\Pi = \left\{ \begin{array}{l} good\_price(acme) \\ in\_fusion(acme, steel) \\ strong(steel) \end{array} \right\},$$

and

$$\Delta = \left\{ \begin{array}{l} buy\_stock(X) \prec good\_price(X) \\ \sim buy\_stock(X) \prec good\_price(X), risky\_company(X) \\ risky\_company(X) \prec in\_fusion(X, Y) \\ risky\_company(X) \prec closing(X) \\ \sim risky\_company(X) \prec in\_fusion(X, Y), strong(Y) \end{array} \right\}.$$

In this case, it can be shown that the answer for $buy\_stock(acme)$ is *Yes* (see García and Simari (2004, Example 2.4) for details).

Example 3.4 (taken from Simari and Loui (1992))**.** Consider the DeLP program $\mathcal{P}_{3.4} = (\Pi, \Delta)$, with

$$\Pi = \left\{ \begin{array}{l} elephant(X) \leftarrow royal\_elephant(X) \\ elephant(X) \leftarrow african\_elephant(X) \\ royal\_elephant(clyde) \\ african\_elephant(clyde) \end{array} \right\},$$

and

$$\Delta = \left\{ \begin{array}{l} gray(X) \prec elephant(X) \\ \sim gray(X) \prec royal\_elephant(X) \end{array} \right\}.$$

This example deals with "on-path versus off-path preemption" in the context of inheritance reasoners. Among the arguments that can be built from $\mathcal{P}_{3.4}$, $\langle \{\sim gray(clyde) \prec elephant(clyde)\}, \sim gray(clyde) \rangle$ is the most

specific; therefore, the answer for the query *gray*(*clyde*) is *No* (Simari and Loui 1992, p. 148).

Example 3.5 (taken from Simari and Loui (1992))**.** Simari and Loui (1992, p. 149) present this example that deals with "defeasible specificity." Consider the DeLP program $\mathcal{P}_{3.5} = (\Pi, \Delta)$, where

$$\Pi = \left\{ \begin{array}{l} adult(ken) \\ university\_student(ken) \end{array} \right\},$$

and

$$\Delta = \left\{ \begin{array}{l} worker(X) \prec adult(X) \\ \sim worker(X) \prec university\_student(X) \\ adult(X) \prec university\_student(X) \end{array} \right\}.$$

DeLP is not able to decide whether or not Ken is a worker as the answer for the query *worker*(*ken*) is *Undecided*.

Example 3.6. The following DeLP program $\mathcal{P}_{3.6} = (\Pi, \Delta)$ codifies the ontology considered in Example 2.6, where

$$\Pi = \left\{ \begin{array}{l} woman(X) \leftarrow married\_woman(X) \\ had\_husband(X) \leftarrow divorcee(X) \\ married\_woman(flor) \\ divorcee(leticia) \end{array} \right\},$$

and

$$\Delta = \left\{ \begin{array}{l} \sim divorcee(X) \prec married\_woman(X) \\ \sim has\_husband(X) \prec had\_husband(X) \\ married\_woman(X) \prec has\_husband(X) \\ married\_woman(X) \prec had\_husband(X) \end{array} \right\}.$$

From this program, the answer for query *woman*(*flor*) is *Yes*, while the answer for the query *divorcee*(*flor*) is *No*, which are correct. However, the answer for the query *has_husband*(*flor*) is *Undecided* as it is not possible to build an argument for such literal. The answer for *woman*(*leticia*) is *Yes* and the answer for *has_husband*(*leticia*) is *No*.

In the next section, we will see under which constraints the translation of arbitrary DL ontologies to DeLP can be performed.

EXPRESSING DL ONTOLOGIES IN DELP

In the presence of inconsistent ontologies, traditional DL reasoners (such as Racer (Haarslev and 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. However, the knowledge engineer is not always available and in some cases, such as when dealing with imported ontologies, he has neither the authority nor the expertise to correct the source of inconsistency. Therefore, 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 by translating DL ontologies into DeLP programs. By doing so we gain the capability of reasoning with inconsistent ontologies. However, we also lose some expressiveness in the involved ontologies. As we will show in Definition 4.1, certain restrictions will have to be imposed on DL ontologies in order to be expressed in the DeLP language.

Our proposal is based in part in the work of (Grosof et al. 2003; Volz 2004) who show that the processing of ontologies can be improved by the use of techniques from the area of logic programming. In particular they have identified a subset of DL languages that can be effectively mapped into a horn-clause logics. Given a DL ontology $\Sigma = (T, A)$, we will consider the Tbox T as partitioned into two disjoint sets—a strict terminology T_S and a defeasible terminology T_D—such that $T = T_S \cup T_D$ and $T_S \cap T_D = \emptyset$.

Considering the analogies between the reasoning systems presented in Figures 1 and 2, we therefore propose translating the DL ontology Σ into a DeLP program $\mathcal{P} = (\Pi, \Delta) = \mathcal{T}(\Sigma)$ by means of a mapping $\mathcal{T}$ from the DL language to the DeLP language. 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. This translation will be achieved by two specialized functions $\mathcal{T}_\Pi$ and $\mathcal{T}_\Delta$, where $\mathcal{T}_\Pi$ translates from a set of DL sentences into a set of DeLP strict rules and $\mathcal{T}_\Delta$ translates from a set of DL sentences into a set of DeLP defeasible rules, such that $\Pi = \mathcal{T}_\Pi(T_D) \cup \mathcal{T}_\Pi(A)$ and $\Delta = \mathcal{T}_\Delta(\Delta)$.

In the rest of this section, we will explain how to achieve the translation of DL ontologies into DeLP programs. For clarity, in spite of being incorrect according to the DeLP syntax conventions, strict rules of the form "$H \leftarrow B_1, \ldots, B_n$" will sometimes be written as "$H \leftarrow B_1 \wedge \cdots \wedge B_n$" and defeasible rules "$H \prec B_1, \ldots, B_n$" as "$H \prec B_1 \wedge \cdots \wedge B_n$." 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 4.1 ($\mathcal{L}_h, \mathcal{L}_b$, and $\mathcal{L}_{hb}$ languages/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 (for details, see the definition of the NNF function in Krötzch, Rudolph, and Hitzler (2007)). Besides, occurrences of $\bot$ and $\top$ are not considered as they can be suitably eliminated as shown in Volz (2004, Table 4.1). These transformations of course do not change the semantics of a terminology.

First we show how to map DL axioms into defeasible rules. As previously mentioned, 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 4.2 ($\mathcal{T}_\Delta$ 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 Figure 3. Besides, intermediate transformations that will end as rules of the form "$(H_1 \wedge H_2) \prec B$" will be rewritten as two rules "$H_1 \prec B$" and "$H_2 \prec B$" (as this is an incorrect DeLP syntax). Similarly, transformations of the form "$H_1 H_2 \prec B$" will be rewritten as "$H_1 \prec B \wedge H_2$," and transformations of the form "$H \prec (B_1 \vee B_2)$" will be rewritten as two rules "$H \prec B_1$" and "$H \prec B_2$."

Example 4.1. Consider the DL terminology

$$T_D = \left\{ \begin{array}{l} \mathsf{Bird} \sqsubseteq \mathsf{Fly} \\ \mathsf{Chicken} \sqsubseteq \neg\mathsf{Fly} \\ \mathsf{Chicken} \sqcap \mathsf{Scared} \sqsubseteq \mathsf{Fly} \end{array} \right\}$$

which 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

$$\Delta = \left\{ \begin{array}{l} \mathit{fly}(X) \prec \mathit{bird}(X) \\ \sim\mathit{fly}(X) \prec \mathit{chicken}(X) \\ \mathit{fly}(X) \prec \mathit{chicken}(X), \mathit{scared}(X) \end{array} \right\}.$$

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

FIGURE 3 Mapping from DL ontologies to DeLP defeasible rules.

Example 4.2 (adapted from Grosof et al. (2003)). Consider the DL axiom:

$$\mathsf{A} \sqcap \exists \mathsf{r}.\mathsf{C} \sqsubseteq \mathsf{B} \sqcap \forall \mathsf{p}.\mathsf{D}.$$

It would be expressed as the (mal-formed) defeasible rule:

$$(b(X) \wedge (d(Z) \prec p(X,Z))) \prec (a(X) \wedge r(X,Y) \wedge c(X)),$$

which is rewritten as the pair of defeasible rules:

$$b(X) \prec a(X), r(X,Y), c(X),$$
$$d(Z) \prec a(X), r(X,Y), c(X), p(X,Z).$$

Next, we present a mapping from DL axioms to strict rules. We are going to assume that strict terminologies are consistent.

Definition 4.3 ($\mathcal{T}^*_\Pi$ mapping from DL sentences to DeLP strict rules). Let A, C, D be concepts, X, Y variables, P, Q properties. The $\mathcal{T}^*_\Pi$: $2^{\mathcal{L}_{DL}} \rightarrow 2^{\mathcal{L}_{DeLP_\Pi}}$ mapping is defined in Figure 4. Besides, intermediate

$$
\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,\ldots,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 4 Mapping from DL ontologies to DeLP strict rules.

transformations of the form "$(H_1 \wedge H_2) \leftarrow B$" will be rewritten as two rules "$H_1 \leftarrow B$" and "$H_2 \leftarrow B$." Similarly, transformations of the form "$H_1 \leftarrow H_2 \leftarrow B$" will be rewritten as "$H_1 \leftarrow B \wedge H_2$," and rules of the form "$H \leftarrow (B_1 \vee B_2)$" will be rewritten as two rules "$H \leftarrow B_1$" and "$H \leftarrow B_2$."

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 "$\sim C(a)$" from "$\sim D(a)$". Thus given "$C_1 \sqcap C_2 \sqcap \cdots \sqcap C_{n-1} \sqcap C_n \sqsubseteq D$," instead of only including the strict rule "$D(X) \leftarrow C_1(X), C_2(X), \ldots, C_{n-1}(X), C_n(X)$" in its translation, we propose including all of its transposes.

Definition 4.4 (transposes of a strict rule)**.** Let $r = H \leftarrow B_1, B_2, B_3, \ldots, B_{n-1}, B_n$ be a DeLP strict rule. The *set of transposes of rule* r, noted as

"*Trans*(r)," is defined as

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

Definition 4.5 ($\mathcal{T}_\Pi$ mapping from DL sentences to DeLP strict rules)**.** We define the mapping from DL ontologies into DeLP strict rules as $\mathcal{T}_\Pi(T) = Trans(\mathcal{T}^*_\Pi(T))$.

Notice that reasoning with transposed strict rules is not more computationally expensive than reasoning without them. Observe that even though as this seems computationally expensive, it is only as expensive as deriving $\sim b$ from $a \leftarrow b$ and $\sim a$.

Also notice that following trends in nonmonotonic reasoning, we do not consider transposition of defeasible rules (Brewka, Dix, and Konolige 1997; Caminada 2008). In this respect, but in the context of default logic, Brewka et al. (1997, p. 45) say that:

> Default logic expressive power is due mainly to the representation of defaults as (nonstandard) inference rules. This representation avoids problems with contraposition of defaults. In classical logic, an implication $A \supset B$ is equivalent to its contraposition $\neg B \supset \neg A$. For defaults, contraposition is sometimes unwanted. For instance, one might believe that computer scientists typically do not know much about nonmonotonic reasoning. From this belief it certainly does not follow that those who know much about nonmonotonic reasoning are typically not computer scientists.

The reader should also notice that the price paid for translating DL ontologies into DeLP programs is the loss of some expressiveness. For instance, as noted by Grosof et al. (2003), DL axioms that generate a rule with a disjunction in its head cannot be represented in logic programming.

DELP-BASED ONTOLOGIES

As previously mentioned, 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 as a set of defeasible inclusion and equality axioms. These axioms are to be considered as an initial, tentative attempt at exploring the problem. Thus, in the presence of inconsistency, to determine the epistemic status of a sentence about some individual's membership in a concept description, a dialectical analysis will be carried out considering all arguments in favor and against its membership.

Knowledge Representation: δ-Ontologies

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

Definition 5.1 (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 $C \sqsubseteq D$, $A \equiv B$, $\top \sqsubseteq \forall P.D$, $\top \sqsubseteq \forall P^{-}.D$, $P \sqsubseteq Q$, $P \equiv Q$, $P \equiv Q^{-}$, or $P^{+} \sqsubseteq P$ 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).

In what follows we assume that the knowledge engineer determines how to encode ontologies in terms of defeasible and strict axioms. Establishing such distinction automatically is a well-known open problem and we discuss some approaches to it in a later section.

Example 5.1. Let $\Sigma_{5.1} = (T_S, T_D, A)$ be a δ-ontology, where

$$T_S = \left\{ \begin{array}{l} \mathsf{Chicken} \sqcup \mathsf{Penguin} \sqsubseteq \mathsf{Bird} \\ \mathsf{Penguin} \sqsubseteq \neg\mathsf{Fly} \end{array} \right\};$$

$$T_D = \left\{ \begin{array}{l} \mathsf{Bird} \sqsubseteq \mathsf{Fly} \\ \mathsf{Chicken} \sqsubseteq \neg\mathsf{Fly} \\ \mathsf{Chicken} \sqcap \mathsf{Scared} \sqsubseteq \mathsf{Fly} \\ \mathsf{Fly} \sqsubseteq \mathsf{Nest\_In\_Trees} \end{array} \right\},$$

and

$$A = \left\{ \begin{array}{l} \mathsf{TINA} : \mathsf{Chicken}, \\ \mathsf{TWEETY} : \mathsf{Penguin}, \\ \mathsf{TINA} : \mathsf{Scared} \end{array} \right\}.$$

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.[3]

The Sbox will be interpreted as a set of strict rules, the Abox as a set of facts, and the Dbox as a set of defeasible rules.

Semantic Interpretation of δ-Ontologies as DeLP Programs

The traditional approach to reasoning in DLs is based on model-theoretic semantics. As DLs are a subset of first-order logic (FOL), entailment has an explosive effect in the presence of inconsistent ontologies (i.e., inconsistency allows to derive all well-formed formulas in the theory). In this work, we propose an argumentative approach to reasoning with inconsistent ontologies. Thus a δ-ontology will be interpreted as a DeLP program. We are assuming that under a traditional DL interpretation, the set $T_S \cup A$ has a model; provided that is the case, then, as required by the DeLP framework, the set $\mathcal{T}_\Pi(T_S) \cup \mathcal{T}_\Pi(A)$ will have a model under a standard logic programming interpretation as well.

Definition 5.2 (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 5.2 (continues Ex. 5.1)**.** Consider again the δ-ontology $\Sigma_{5.1}$. $\Sigma_{5.1}$ is interpreted as the DeLP program $\mathcal{P}_{5.1} = (\Pi, \Delta)$, where

$$\Pi = \left\{ \begin{array}{l} bird(X) \leftarrow chicken(X) \\ \sim chicken(X) \leftarrow \sim bird(X) \\ bird(X) \leftarrow penguin(X) \\ \sim penguin(X) \leftarrow \sim bird(X) \\ \sim fly(X) \leftarrow penguin(X) \\ \sim penguin(X) \leftarrow fly(X) \\ chicken(tina) \\ penguin(tweety) \\ scared(tina) \end{array} \right\},$$

and

$$\Delta = \left\{ \begin{array}{l} fly(X) \prec bird(X) \\ \sim fly(X) \prec chicken(X) \\ fly(X) \prec chicken(X), scared(X) \\ nest\_in\_trees(X) \prec fly(X) \end{array} \right\}.$$

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. 3.2, we must enforce some internal coherence between the Abox and the Tbox.

Definition 5.3 (internal coherence in Aboxes. Consistency of Aboxes w.r.t. Sboxes)**.** Let $\Sigma = (T_S, T_D, A)$ be a δ-ontology. The Abox A is *internally coherent* iff there are no pair of assertions $a : C$ and $a : \neg C$, for any individual a and any class 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 5.3. Let $\Sigma_{5.3} = (T_S, \emptyset, A)$ be a δ-ontology such that $T_S = \{(\mathsf{C} \sqsubseteq \mathsf{D}), (\mathsf{D} \sqsubseteq \neg\mathsf{F})\}$ and $A = \{(\mathsf{B} : \mathsf{C}), (\mathsf{B} : \mathsf{F})\}$. $\Sigma_{5.3}$ is expressed as $\mathcal{P}_{5.3} = (\Pi, \emptyset)$, where $\Pi = \mathcal{T}_\Pi(T_S) \cup \mathcal{T}_\Pi(A) = \{c(b), c(b), (d(X) \leftarrow c(X)), (\sim c(X) \leftarrow \sim d(X)), (\sim f(X) \leftarrow d(X)), (\sim d(X) \leftarrow f(X))\}$, from which it is possible to strictly derive $f(b)$ and $\sim f(b)$. Therefore A is not consistent w.r.t. T_S.

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 5.4 (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 $\mathcal{P} = (\mathcal{T}_\Pi(T_S) \cup \mathcal{T}_\Pi(A), \mathcal{T}_\Delta(T_D))$.

1. 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. $\mathcal{P}$.
2. The *individual a justifiedly belongs to class C* (noted as "C_w^a") iff there exists a warranted argument $\langle \mathcal{A}, C(a) \rangle$ w.r.t. $\mathcal{P}$.
3. 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. $\mathcal{P}$.

We have the following straightforward result.

Property 5.1. Given a δ-ontology Σ, a concept name C, and an individual a, C_s^a implies C_w^a, and C_w^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.

The next example remarks the way our proposal is capable of handling instance checking under the *Open World Assumption* assumed by Semantic Web standards.

Example 5.4. Consider the δ-ontology $\Sigma_{5.4} = (\emptyset, \emptyset, \{\mathsf{TWEETY} : \mathsf{Bird}\})$, that is interpreted as the DeLP program $\mathcal{P}_{5.4} = (\{bird(tweety)\}, \emptyset)$. It holds that $\mathsf{Bird}_w^{\mathsf{TWEETY}}$ as the argument $\langle \emptyset, bird(tweety)\rangle$ is trivially warranted w.r.t. $\mathcal{P}_{5.4}$. However it holds neither $\mathsf{Bird}_w^{\mathsf{OPUS}}$ nor $\neg\mathsf{Bird}_w^{\mathsf{OPUS}}$ as the answer for $bird(opus)$ is Undecided because no argument for that literal (or its negation) can be built from $\mathcal{P}_{5.4}$.

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

Definition 5.5 (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 (resp. justified) membership* of a to a complex concept is defined as:

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

Notice that in the above definition the concept E can be actually larger than the complex concept it represents and it is introduced solely for expressing the initial DeLP query.

Property 5.2. Let $\Sigma = (T_S, T_D, A)$ be a δ-ontology. It cannot be the case that an individual a belongs justifiedly to concept C and $\neg C$ simultaneously.

Proof. Suppose that both C^a_w and $(\neg C)^a_w$. 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 is impossible as DeLP cannot warrant two complementary literals simultaneously (as shown in Property 3.1).

Property 5.3. Let $\Sigma = (T_S, T_D, A)$ be a δ-ontology. It cannot be the case an individual a belongs strictly to concept C and $\neg C$ simultaneously.

Proof. Suppose that C^a_s and $(\neg C)^a_s$, then there must exist two arguments $\langle \emptyset, C(a) \rangle$ and $\langle \emptyset, \sim C(a) \rangle$ w.r.t. $(\mathcal{T}_\Pi(T_S) \cup \mathcal{T}_\Pi(A), \mathcal{T}_\Delta(T_D))$. Therefore $\mathcal{T}_\Pi(T_S) \cup \mathcal{T}_\Pi(A)$ will be inconsistent contradicting Definition 5.3.

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 in knowing all 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 that 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 $(\mathcal{T}_\Pi(T_S) \cup \mathcal{T}_\Pi(A), \mathcal{T}_\Delta(T_D))$ (see Figure 5).
- *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 $(\mathcal{T}_\Pi(T_S) \cup \mathcal{T}_\Pi(A), \mathcal{T}_\Delta(T_D))$ (see Figure 6).

Property 5.4. The running time of the processes for "open retrieval" and "retrieval of all classes" is finite.

```
function Retrieval( Σ = (T_S, T_D, A); c : Concept ) : Set of Individual
var S : Set of Individual
begin
        I := Set of individuals appearing in Σ = (T_S, T_D, A)
        S := ∅
        for each individual name a ∈ I do
            if ⟨A, c(a)⟩ is warranted then
                S := S ∪ {a}
            end if
        end for
        return S
end
```

FIGURE 5 Open retrieval.

```
function RetrievalOfAllClasses( Σ = (T_S, T_D, A); a : Individual ) : Set of Class
var S : Set of Class
begin
        C := Set of concepts appearing in Σ = (T_S, T_D, A)
        S := ∅
        for each concept name c ∈ C do
            if ⟨A, c(a)⟩ is warranted then
                S := S ∪ {c}
            end if
        end for
        return S
end
```

FIGURE 6 Retrieval of all classes.

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, Fillottrani, and Simari (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 and Simari 2004), then processes for "open retrieval" and for "retrieval of all classes" always terminate.

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 a 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 6.1 (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 concept or role name 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 $q_\mathcal{G}$ and $q_{\mathcal{S}_i}$ are queries of the same arity defined over the 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 source 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 6.1. Consider the *global* δ-ontology $\mathcal{G} = (\emptyset, T_D^\mathcal{G}, \emptyset)$ presented in Figure 7. 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.

Notice that the terminology $T_D^\mathcal{G}$ expresses a conflict w.r.t. the concept "Good". If we find that some individual in the Abox can be proven to be member of both concepts "Swimmer" and "Greek", then the Abox would be incoherent from a traditional DL point of view because that individual

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 7 Global ontology $\mathcal{G} = (\emptyset, T_D^\mathcal{G}, \emptyset)$.

would belong both to "Good" and to "¬Good, indicating that the concept "Good" should be empty and having one individual at the same time. We will show how this type of situation can be handled naturally in DeLP.

Example 6.2 (continues Ex. 6.1)**.** In Figure 8, we present two source ontologies: $\mathcal{S}_1$ about water activities, and $\mathcal{S}_2$ on computer programming. In local source 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 local source 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

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

Defeasible terminology $T_D^{\mathcal{S}_1}$**:**

Free_Diver ⊔ Scuba_Diver ⊑ Diver
Saturation_Diver ⊑ Scuba_Diver
∃swims.stroke ⊑ Swimmer

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

CRAWL : stroke; SIDE : stroke
⟨JOHN, CRAWL⟩ : swims; ⟨JOHN, SIDE⟩ : 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 ⊔ OOP_Lang ⊑ Lang

Defeasible terminology $T_D^{\mathcal{S}_2}$**:**

∃programs.Lang ⊑ Programmer
∃programs.Lang ⊓ Failed_Prog_101 ⊑ ¬Programmer
reads ⊓ writes ⊑ programs

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

PROLOG : LP_Lang; JAVA : OOP_Lang
⟨JOHN, PROLOG⟩ : reads; ⟨JOHN, PROLOG⟩ : writes
⟨MARY, JAVA⟩ : reads; ⟨MARY, JAVA⟩ : writes
⟨PAUL, JAVA⟩ : reads; ⟨PAUL, JAVA⟩ : writes
PAUL : Failed_Prog_101

FIGURE 8 Source ontologies $\mathcal{S}_1$ and $\mathcal{S}_2$.

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.

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 "¬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 in 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); Euzenat and Shvaiko (2007). Moreover, we are going to assume unique name assumption w.r.t. references to individuals in Aboxes through our presentation.

Example 6.3 (continues Ex. 6.2)**.** Consider again global ontology $\mathcal{G}$ and source ontologies $\mathcal{S}_1$ and $\mathcal{S}_2$. Articulation of definitions in $\mathcal{G}$ with those in $\mathcal{S}_1$ and $\mathcal{S}_2$ is done by bridge ontologies $\mathcal{M}_1$ and $\mathcal{M}_2$, resp. (see Figure 9). For clarity, in bridge ontologies we tag concept and property names with their defining ontology name.

Bridge ontology $\mathcal{M}_1$ expresses that role "swims" in $\mathcal{S}_1$ corresponds to role "can_swim" in $\mathcal{G}$; concept "Diver" in $\mathcal{S}_1$ refers to "Diver" in $\mathcal{G}$; the concept (anonymously 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

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 9 Bridge ontologies $\mathcal{M}_1$ and $\mathcal{M}_2$.

to "Race_Stroke" in $\mathcal{G}$. Bridge ontology $\mathcal{M}_2$ indicates that the concept "Programmer" in $\mathcal{S}_2$ corresponds to the concept "Greek" defined in $\mathcal{G}$.

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 way 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 a DeLP program.

Definition 6.2 (interpretation of an ontology integration system)**.** Let $\mathcal{I} = (\mathcal{G}, \mathcal{S}, \mathcal{M})$ be an ontology integration system such that $\mathcal{S} = \{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$ and $\mathcal{M} = \{\mathcal{M}_1, \ldots, \mathcal{M}_n\}$, 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 $i = 1, \ldots, n$,
- $\mathcal{M}_i = (T_S^{\mathcal{M}_i}, T_D^{\mathcal{M}_i})$, with $i = 1, \ldots, n$.

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

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

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

Example 6.4 (continues Ex. 6.3)**.** 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 Figures 10–12, 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 $\Delta_{\mathcal{M}_2}$.

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

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

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

FIGURE 10 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:

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

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:

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

FIGURE 11 DeLP programs $\mathcal{P}_1$ and $\mathcal{P}_2$ obtained from source ontologies $\mathcal{S}_1$ and $\mathcal{S}_2$, resp.

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

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.

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 12 Bridge ontologies expressed as defeasible rules.

Definition 6.3 (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 $\mathcal{G}$.

1. Individual *a potentially belongs* to concept C iff there exists an argument $\mathcal{A}$ for the literal $C(a)$ w.r.t. DeLP program $\mathcal{I}_{DeLP}$.
2. Individual *a justifiedly belongs* to concept C iff there exists a warranted argument $\mathcal{A}$ for the literal $C(a)$ w.r.t. DeLP program $\mathcal{I}_{DeLP}$.
3. Individual *a strictly belongs* 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 and Loui 1992; Stolzenburg et al. 2003) as the criterion for argument comparison. This is just an example as other criteria could be used (e.g., (Ferreti, Errecalde, García, and Simari 2007)).

Example 6.5 (continues Ex. 6.4)**.** Consider again the 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. 6.3, it is necessary to determine if there exist warranting 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. We will see that as John is both a geek and a swimmer, it will not be possible to determine if he is good at sports or not. 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 the concept "Good." Formally, there exists an argument $\langle \mathcal{A}_1, good(john) \rangle$ where

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

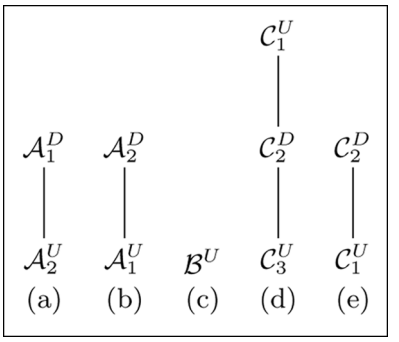

FIGURE 13 Dialectical trees for queries *good*(*john*), *good*(*mary*), and *good*(*paul*).

However, John belongs potentially to the 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

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

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 Figure 13(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 Figure 13.(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 Figure 13.(c)), where

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

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), with

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

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

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

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

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

Thus, the dialectical tree for the query "$good(paul)$" has three nodes (see Figure 13.(d)). With respect to the query "$\sim good(paul)$" for determining if Paul belongs justifiedly to the concept "¬Good," argument $\mathcal{C}_2$ is defeated by argument $\mathcal{C}_1$ (see Figure 13.(e)). Therefore, the answer to the query "good(paul)" is YES, and we conclude that Paul belongs justifiedly to the concept "Good."

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 nonstandard 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 nonstandard inference relation, Huang et al. (2005) propose using an alternative classification to distinguish answers to queries.

Definition 7.1 (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. *Overdetermined*: $\Sigma \mid\!\approx \phi$ 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. 5.4), $\Sigma \mid\!\approx \phi$ corresponds to a YES answer to query ϕ w.r.t. program $\mathcal{T}(\Sigma)$.

Property 7.1. 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 overdetermined.

Proof. Suppose, on the contrary that the answer to ϕ is overdetermined. 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 as it would contradict Property 3.1.

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 $\sim Q$. Notice also that according to Volz (2004), the transformation from DL to logic programming is sound but not complete.

Definition 7.2 (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 7.2. $\mid\!\approx$ 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. 3.2) is consistent and as $\mathcal{T}$ is a transformation that preserves semantics Grosof et al. (2003), there must exist a subset $\Sigma' \subseteq \Sigma$ such that $\mathcal{T}(\Sigma') = \Pi_S \cup \Pi_A \cup \mathcal{A}$.

Definition 7.3 (consistency (Huang et al. 2005)). An inconsistency reasoner $\approx$ is *consistent* iff $\Sigma \approx \phi \Rightarrow \Sigma \not\approx \neg\phi$.

Property 7.3. $\approx$ is a consistent inconsistency reasoner.

Proof. Corollary of Prop. 7.1.

Definition 7.4 (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 7.4. $\approx$ is a meaningful inconsistency reasoner.

Proof. Trivial from Props. 7.2 and 7.3.

Implementation Issues

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 that DLP languages suffice to express 77–87% of the analyzed ontologies and that they 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 justification abstract machine (JAM) has been specially developed for an efficient implementation of DeLP (García and Simari 2004). The JAM provides an argument-based extension of the traditional Warren's abstract machine (WAM) for PROLOG. A full-fledged implementation of DeLP is available online,[4] including facilities for visualizing arguments and dialectical trees.

When we map DL equality axioms into DeLP rules, 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 can 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, notice this is an intrinsic problem of DeLP implementations based on a PROLOG meta interpreter. A possible solution involves modifying the DeLP interpreter such that it keeps track of the search space of rule instantiations into ground rules. To avoid such looping situations, every time a new rule is

about to be instantiated, the modified DeLP interpreter needs to check if the rule instance was already computed.

COMPUTING THE PARTITION OF TBOXES IN SBOXES AND DBOXES

One of the advantages of ontology representation languages based on DL is that existing reasoners can be used to compute implicit subsumption relations between classes. As a byproduct, this process is useful to identify unsatisfiable (inconsistent) classes. However, standard DL reasoners based on the tableaux algorithm (such as Racer (Haarslev and Möller 2001), FaCT (Horrocks 1998), Pellet (Parsia and Sirin 2004)), only provide a list of unsatisfiable classes. The process of "debugging" the ontology (i.e., to determine why a reasoner has inferred that a class is unsatisfiable is left to the user). When the knowledge engineer is faced with several unsatisfiable classes on moderately large ontology, this process can become rather difficult (Wang et al. 2005). As it was mentioned in the first section, a notable exception is a recent work of Horridge et al. (2008) that is capable of providing justifications (explanations) of how an entailment holds.

In the approach proposed in this work, problematic axioms in the terminology of an ontology are separated in a distinguished set called Dbox while nonproblematic axioms are conserved in a set called Sbox. In the previous sections, such sets have been considered as given—that is, given a certain terminology T, the knowledge engineer decides somehow how to partition it in order to populate the Sbox T_S and the Dbox T_D. In this section, we discuss how to perform such partition automatically.

Two Naïve Approaches

We now present two simple approaches to the problem of partitioning a Tbox in an Sbox and a Dbox: the first consists of considering all the axioms in the Tbox as defeasible (i.e., all the Tbox axioms are regarded as Dbox axioms); the second consists of grouping all the logic programming rules obtained from the Tbox that have heads with complementary literals.

Definition 8.1 (associate δ-ontology). Let $\Sigma = (T, A)$ be a DL ontology, we define Σ's *associate δ-ontology*, noted $Asoc(\Sigma)$, as the δ-ontology obtained from Σ by applying some partition strategy to T.

Definition 8.2 (associate DeLP program). Given a DL ontology $\Sigma = (T, A)$, we define Σ's *associate DeLP program*, noted $Prog(\Sigma)$, as the DeLP program obtained when translating Σ to DeLP.

Strategy 8.1. Given a DL ontology $\Sigma = (T, A)$, Σ's associate δ-ontology is $Asoc(\Sigma) = (\emptyset, T, A)$; and Σ's associate DeLP program is $Prog(\Sigma) = (\mathcal{T}_\Pi(A), \mathcal{T}_\Delta(T))$.

Example 8.1. Consider the DL ontology $\Sigma_{8.1} = (T, A)$, where: $T = \{(\mathsf{C} \sqsubseteq \mathsf{D}), (\mathsf{E} \sqsubseteq \neg\mathsf{D})\}$ and $A = \{(\mathsf{a} : \mathsf{C}), (\mathsf{a} : \mathsf{E})\}$. In this case, Σ's associate δ-ontology is $Asoc(\Sigma_{8.1}) = (\emptyset, T, A)$ and Σ's associate DeLP program is $Prog(\Sigma_{8.1}) = (\{(d(X) \prec c(X)), (\sim d(X) \prec e(X))\}, \{c(a), e(a)\})$.

Property 8.5. Given a DL ontology $\Sigma = (T, A)$, if the Abox A is coherent, then the set of facts $\mathcal{T}_\Pi(A)$ is consistent.

Strategy 8.2. Let *Convert* be the function that converts a set of strict rules into a set of defeasible rules.[5] Given a DL ontology $\Sigma = (T, A)$, let $\mathcal{P}$ an extended logic program where its set of facts is defined as $\mathcal{T}_\Pi(A)$ and its set of strict rules as $\mathcal{T}^*_\Pi(T)$. Let $Part : \mathcal{L}_{ELP} \to (\mathcal{L}_{DeLP_\Pi}, \mathcal{L}_{DeLP_\Delta})$ be a function that partitions such a set of rules into a subset of strict rules and another subset of defeasible rules. Formally, $Part(R) = (R_\Pi, R_\Delta)$, where

$$R_\Delta =_{df} Convert(r \in R \mid \text{there exists } r' \in R \text{ such that } Head(r) = \overline{Head(r')}),$$
$$R_\Pi =_{df} R - R_\Delta.$$

Example 8.2. Consider the Tbox $T_{8.2}$ with

$$T = \left\{ \begin{array}{l} (\mathsf{A} \sqcap \mathsf{B}) \sqcup \mathsf{C} \sqsubseteq \mathsf{D} \sqcap \mathsf{E} \\ \neg\mathsf{A} \sqcup \mathsf{F} \sqsubseteq \neg\mathsf{E} \end{array} \right\}.$$

From this Tbox, the following set R of rules is generated:

$$R = \left\{ \begin{array}{l} d(X) \leftarrow a(X), b(X) \\ e(X) \leftarrow a(X), b(X) \\ d(X) \leftarrow c(X) \\ e(X) \leftarrow c(X) \\ \sim e(X) \leftarrow \sim a(X) \\ \sim e(X) \leftarrow f(X) \end{array} \right\}.$$

By the application of Strategy 8.2, we obtain the following sets R_Π of strict rules and R_Δ of defeasible rules, where

$$R_\Pi = \left\{ \begin{array}{l} d(X) \leftarrow a(X), b(X) \\ d(X) \leftarrow c(X) \end{array} \right\},$$

and

$$R_{\Delta} = \left\{ \begin{array}{l} e(X) \prec a(X), b(X) \\ e(X) \prec c(X) \\ \sim e(X) \prec \sim a(X) \\ \sim e(X) \prec f(X) \end{array} \right\}.$$

Approach Based on Converting Problematic DL Axioms into Dbox Axioms

We now consider another approach to the problem of partitioning a Tbox in an Sbox and a Dbox automatically. In the previous section, given an inconsistent DL ontology, we worked with the DeLP rules generated by the translation function from DL to DeLP, turning some strict rules into defeasible ones. We propose instead to work directly with DL axioms by separating them into strict and defeasible axioms in order to see how to obtain a δ-ontology to reason with the framework previously presented.

Lam et al. (2006) propose an algorithm that extends Meyer, Lee, Booth, and Pan's (2006) tableaux algorithm. The algorithm not only pinpoints the problematic axioms, but also traces which parts of the axioms are responsible for the unsatisfiability of a target concept C. We present next an example of Lam et al. (2006) that explains how this process can be performed.

Example 8.3 (Lam et al. 2006)**.** Assume that a DL ontology Σ contains the following axioms:

$$r_1 : \mathsf{A} \equiv \mathsf{C} \sqcap \mathsf{D} \sqcap \mathsf{G}$$
$$r_2 : \mathsf{C} \equiv \mathsf{E} \sqcap \mathsf{F}$$
$$r_3 : \mathsf{E} \equiv \neg \mathsf{D} \sqcap (\exists \mathsf{r}.\mathsf{D})$$
$$r_4 : \mathsf{H} \equiv \forall \mathsf{r}.\mathsf{C}$$

It can be shown that the concept A is unsatisfiable, by using standard DL TBox reasoning. Existing approaches either identify the minimally unsatisfiable sub-ontology $\Sigma_{min} = \{r_1, r_2, r_3\}$ or calculate the maximally satisfiable sub-ontologies $\Sigma_{max_1} = \{r_2, r_3, r_4\}$, $\Sigma_{max_2} = \{r_1, r_3, r_4\}$ and $\Sigma_{max_3} = \{r_1, r_2, r_4\}$. In short, one of the axioms r_1, r_2 or r_3 should be removed from Σ. However, it can be shown that we do not need to remove any of the above 'whole' axioms. In fact, we can simply remove any one of the following parts of axioms: (i) $\mathsf{A} \sqsubseteq \mathsf{C}$, (ii) $\mathsf{A} \sqsubseteq \mathsf{D}$, (iii) $\mathsf{C} \sqsubseteq \mathsf{E}$, or (iv) $\mathsf{E} \sqsubseteq \neg \mathsf{D}$, and Σ becomes satisfiable.

Thus, using Lam et al.'s (2006) algorithm as a "black box," we propose the following strategy to partition a DL Tbox into a δ-ontology Sbox and a Dbox.

Definition 8.3 (set of concepts associated to an axiom/terminology)**.** Let $r_1, \ldots, r_n$ a set of DL inclusion axioms such that form a terminology $T = \{r_1, \ldots, r_n\}$. We define $Concepts(r_i)$ as the set of named concepts of axiom r_i. We extend the definition to the set of named concepts in a terminology T as:

$$Concepts(T) = \bigcup_{i=1,\ldots,n} Concepts(r_i).$$

Definition 8.4 (Lam et al.'s algorithm)**.** Let T be a DL terminology and C a named concept in T. We define the function $Lam_C(T)$ as Lam's algorithm that takes as input T and C, and returns as output the set of named concepts in T relevant to the unsatisfiability of C.

Strategy 8.3. Let $\Sigma = (T, A)$ be a DL ontology and C a concept name such that $C \in Concepts(\Sigma)$. The separation function *Sep* of a DL Tbox in a δ-ontology Sbox and Dbox, is defined as $Sep(T) = (T_S, T_D)$ where:

$$T_D = r \in T \mid (Concepts(r) \cap Lam_C(T)) \neq \emptyset,$$

and

$$T_S = T - T_D.$$

Example 8.4. Consider the following DL terminology $T_{8.1}$ (taken from Lam et al. (2006)):

$$T_{8.4} = \left\{ \begin{array}{l} r_1 : \mathsf{Bird}^* \sqsubseteq \mathsf{Flies}^* \sqcap \mathsf{Animal} \\ r_2 : \mathsf{Eagle} \sqsubseteq \mathsf{Bird} \\ r_3 : \mathsf{Penguin}^* \sqsubseteq \mathsf{Bird}^* \sqcap (\neg \mathsf{Fly})^* \end{array} \right\}.$$

Following Lam et al.'s (2006) notation, the concepts labeled with a star are relevant to the unsatisfiability of the concept "Penguin"; therefore,

$$Lam_{\mathsf{Penguin}}(T_{8.4}) = \mathsf{Bird}, \mathsf{Flies}, \mathsf{Penguin}, (\neg \mathsf{Flies}).$$

Thus, instead of just eliminating all the axioms containing at least a concept labeled with an star (i.e., r_1 and r_3), which would cause the loss of some inferences, we propose converting them into Dbox axioms. Hence, the $T_{8.4}$ is partitioned as an Sbox $T_S = \{r_2\}$ and a Dbox $T_D = \{r_1, r_3\}$.

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 (DLPs), 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 and 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 and Vermeir (2002), we allow to perform inferences from inconsistent ontologies by considering subsets (arguments) of the original ontology. Heymans and 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 and Simari 2004).

Eiter, Lukasiewicz, Schindlauer, and Tompits (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 and 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, which support or reject possible

correspondences between ontologies in the context of a multi-agent system. In our work, we assume correspondences between ontologies as given.

Antoniou and 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, Resource Description Framework Schema (RDF(S)) and parts of OWL ontologies. RDF data of the form *rdf*(*Subject*, *Predicate*, *Object*) is translated as Prolog facts of the form *Predicate*(*Subject*, *Object*). They also define Prolog rules for processing RDF Schema information (e.g., C(X):-rdf:type(X,C)) for modeling the type construct). All of the rules are created at compile time before actual querying takes place. To the best of our knowledge, Antoniou and Bikakis's (2007) approach distinguishes between strict and defeasible and possess flexibility for defining different priority relations between arguments. As our approach is based on DeLP, it also distinguishes between strict and defeasible rules, and it also allows to replace the comparison criterion between arguments in a modular way. Antoniou and Bikakis translate the semantics of their argumentation system into Prolog, the semantics of OWL sentences into Prolog, the semantics of RuleML sentences into Prolog and then they perform query processing. We translate OWL into DL and the into DeLP, thus keeping the knowledge representation (in DeLP) separated from the query processing (performed into the JAM).

Horridge et al. (2008) present a justification-based approach to reasoning with ontologies. In that context, a justification for an entailment in an OWL ontology is a minimal subset of the ontology that is sufficient for that entailment to hold. Besides they are able to find laconic justifications that provide minimal axioms supporting an entailment, which can also be used to pinpoint the cause of inconsistencies. The notion of justification for an entailment in Horridge et al. (2008) is similar to the notion of argument for a literal in our work as an argument is made up of a minimal subset of the defeasible information in a DeLP program along with the strict information that allows to derive a defeasible conclusion (see Def. 3.2), thus providing a sort of justification for an entailment to hold. However, in the case of inconsistencies, DeLP is not only able to consider all the arguments for a literal, but also the defeat relation holding among those arguments in order to determine what the valid consequences of a δ-ontology are (characterized as the warranted arguments). Similarly, to axiom minimization in laconic justifications, the DeLP rules interpreting a δ-ontology can be considered as minimal versions of DL axioms (see Definitions 4.2 and 4.3) as they are horn-rules where disjunctions in the body and conjunctions in the head trigger the proliferation of rules. For example, in Horridge et al. (2008), the inclusion axiom $\mathsf{A} \sqcap \mathsf{B} \sqsubseteq \mathsf{C} \sqcap \mathsf{D}$ is expressed as two (smaller) axioms

$\mathsf{A} \sqcap \mathsf{B} \sqsubseteq \mathsf{C}$ and $\mathsf{A} \sqcap \mathsf{B} \sqsubseteq \mathsf{D}$; DeLP represents the former axiom as two rules $c(X) \prec a(X), b(X)$ and $d(X) \prec a(X), b(X)$, thus providing a *de facto* minimization.

Horridge et al. (2008, Sect. 3) consider two cases for motivating fine-grained justifications: internal and external masking. When internal masking occurs, Horridge et al. (2008) present the ontology $\Sigma = \{\mathsf{B} \sqsubseteq \neg\mathsf{C} \sqcap \mathsf{D}, \mathsf{B} \sqsubseteq \neg\mathsf{C}\}$, where the concept B is unsatisfiable for two distinct reasons. We show next how our approach can detect the pairs of conflicting arguments depicting that situation. In our framework, Σ along with an Abox assertion $\mathsf{a} : \mathsf{B}$ is interpreted as the DeLP program $\{(\sim c(X) \prec b(X)), (d(X) \prec b(X)), (c(X) \prec b(X)), (\sim d(X) \prec b(X)), b(a)\}$ from where the following arguments can be found: $\langle\{c(a) \prec b(a)\}, c(a)\rangle$, $\langle\{\sim c(a) \prec b(a)\}, \sim c(a)\rangle$, $\langle\{d(a) \prec b(a)\}, d(a)\rangle$, and $\langle\{\sim d(a) \prec b(a)\}, \sim d(a)\rangle$, indicating the presence of inconsistency and showing that $\mathsf{C}^{\mathsf{a}}_{p}$ and $\neg\mathsf{C}^{\mathsf{a}}_{p}$ along with $\mathsf{D}^{\mathsf{a}}_{p}$ and $\neg\mathsf{D}^{\mathsf{a}}_{p}$. In spite of this, our framework will not be able to determine the justified membership of a to neither C nor D as the DeLP answers for the queries $c(a)$ is $d(a)$ undecided. Likewise, in the case of external masking, Horridge et al. (2008) present the ontology $\Sigma' = \{\mathsf{B} \sqsubseteq \mathsf{D} \sqcap \neg\mathsf{D} \sqcap \mathsf{C}, \mathsf{B} \sqsubseteq \neg\mathsf{C}\}$, where the concept B is unsatisfiable but there is no justification for that fact. Our framework interprets Σ' as the DeLP program $\{(d(X) \prec b(X)), (\sim d(X) \prec b(X)), (c(X) \prec b(X)), (\sim c(X) \prec b(X)), b(a)\}$ from where, as in the former case, there are arguments for the literals $c(a)$, $\sim c(a)$, $d(a)$ and $\sim d(a)$ indicating the source of inconsistency, yet the answer for the justified membership of individual a to the concepts C or D is undecided.

CONCLUSIONS

We have presented a framework for reasoning with inconsistent DL ontologies. Our proposal involves expressing a DL ontology in terms of a DeLP program by means of a translation function $\mathcal{T}$. This resulted in the characterization of δ-ontologies, encoded using strict and defeasible axioms. We also provided a semantic interpretation of δ-ontologies as DeLP programs, formalizing two major inference tasks associated with a traditional DL setting, namely, instance checking and retrieval. Given a query ϕ and an inconsistent ontology Σ, these inference tasks rely on a dialectical analysis to be performed on the DeLP program $\mathcal{T}(\Sigma)$, where all arguments in favor and against ϕ's acceptance are taken into account. In this setting, the notion of class membership was suitably extended to distinguish among potential, justified, and strict membership of an individual to a class.

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 solution to this problem could be taken in that direction using some suitable extension of disjunctive logic programming (Terracina, Francesco, Panetta, and Leone 2008; Ricca and Leone 2007). Part of our current research work is oriented in this direction.

REFERENCES

Antoniou, G., and A. Bikakis. 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.

Antoniou, G., D. Billington, and M. Maher. 1998. Normal forms for defeasible logic. In: *Proceedings of International Joint Conference and Symposium on Logic Programming*, 160–174. Boston: MIT Press.

Antoniou, G., M. J. Maher, and D. Billington. 2000. Defeasible logic versus logic programming without negation as failure. *Journal of Logic Programming* 42:47–57.

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

Bench-Capon, T. J. M., and P. E. Dunne. 2007. Argumentation in artificial intelligence. *Artif. Intell.* 171(10–15):619–641.

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

Brena, R., C. Chesñevar, and J. Aguirre. 2006. Argumentation-supported information distribution in a multiagent system for knowledge management. In: *Proc. ArgMAS 2005* (Utrecht, The Netherlands, July 2005). *LNCS* 4049, 279–296, Springer-Verlag.

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

Calvanese, D., G. D. Giacomo, and M. Lenzerini. 2001. A framework for ontology integration. In: *Proc. 1st Semantic Web Working Symposium (SWWS 2001)*, 303–316.

Caminada, M. 2008. On the issue of contraposition of defeasible rules. In: *Frontiers in Artificial Intelligence and Applications*, eds. P. Besnard, S. Doutre, and A. Hunter, Vol. 172, 109–115. 'COMMA', IOS Press.

Carbogim, D., D. Robertson, and J. Lee. 2000. Argument-based applications to knowledge engineering. *The Knowledge Engineering Review* 15(2):119–149.

Cecchi, L. A., P. R. Fillottrani, and G. R. Simari. 2006. On complexity of DeLP through game semantics. In: *11th. Intl. Workshop on Nonmonotonic Reasoning*, eds. J. Dix and A. Hunter, 386–394.

Chesñevar, C., R. Brena, and J. Aguirre. 2005a. Knowledge distribution in large organizations using defeasible logic programming. In: *Proc. 18th Canadian Conference on AI LNCS*, Vol. 3501, 244–256. Springer-Verlag.

Chesñevar, C., R. Brena, and J. Aguirre. 2005b. Modelling power and trust for knowledge distribution: an argumentative approach. *LNAI* Springer Series *(Proc. 3rd Mexican International Conference on Artificial Intelligence – MICAI 2005)* 3789:98–108.

Chesñevar, C. I., A. G. Maguitman, and G. R. Simari. 2006. Argument-based critics and recommenders: A qualitative perspective on user support systems. *Data Knowl. Eng.* 59(2): 293–319.

Chesñevar, C. I., A. Maguitman, and R. Loui. 2000. Logical models of argument. *ACM Computing Surveys* 32(4):337–383.

Chesñevar, C., and A. Maguitman. 2004a. An argumentative approach to assessing natural language usage based on the web corpus. In: *Proc. 16th ECAI Conf.*, 581–585. Valencia, Spain.

Chesñevar, C., and A. Maguitman. 2004b. ArgueNet: An argument-based recommender system for solving web search queries. In: *Proc. 2nd IEEE Intl. IS-2004 Conference*, 282–287. Varna, Bulgaria.

Chesñevar, C., G. Simari, T. Alsinet, and L. Godo. 2004. A logic programming framework for possibilistic argumentation with vague knowledge. In: *Proc. Intl. Conference in Uncertainty in Artificial Intelligence (UAI 2004)*, 76–84. Banff, Canada.

Chesñevar, C., G. Simari, L. Godo, and T. Alsinet. 2005. Argument-based expansion operators in possibilistic defeasible logic programming: Characterization and logical properties. *LNAI/LNCS Springer Series*, Vol. 3571 (*Proc. 8th ECSQARU Intl. Conference*, 353–365. Barcelona, Spain).

Dimopoulos, Y., and A. Kakas. 1995. Logic programming without negation as failure. In: *Logic Programming*, ed. J. Lloyd, 369–383. Cambridge, MA: MIT Press.

Eiter, T., T. Lukasiewicz, R. Schindlauer, and H. Tompits. 2004. Combining answer set programming with description logics for the semantic web. *KR 2004*, 141–151.

Euzenat, J., and P. Shvaiko. 2007. *Ontology Matching*. Berlin/Heidelberg: Springer-Verlag.

Ferreti, E., M. Errecalde, A. J. García, and G. R. Simari. 2007. An application of defeasible logic programming to decision making in a robotic environment. In: *Ninth International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR'07*.

García, A. J., and G. R. Simari. 2004. Defeasible logic programming: An argumentative approach. *Theory and Practice of Logic Programming* 4(1):95–138.

Geffner, H., and J. Pearl. 1990. A framework for reasoning with defaults. In: *Knowledge Representation and Defeasible Reasoning*, eds. R. L. H. E. Kyburg, and G. Carlson, 245–265. London: Kluwer Academic Publishers.

Gómez, S. A., C. I. Chesñevar, and G. R. Simari. 2006. An approach to handling inconsistent ontology definitions based on the translation of description logics into defeasible logic programming. In: *Procs. XII Argentinian Conference in Computer Science (CACIC'06)*, 1185–1196.

Gómez, S. A., C. I. Chesñevar, and G. R. Simari. 2008a. An argumentative approach to reasoning with inconsistent ontologies. In: *Proc. Knowledge Representation in Ontologies Workshop (KROW 2008)*, eds. T. Meyer and M. A. Orgun, Vol. CPRIT 90, 11–20. Sydney, Australia.

Gómez, S. A., C. I. Chesñevar, and G. R. Simari. 2008b. Defeasible reasoning in web forms through argumentation. *International Journal of Information Technology & Decision Making* 7:71–101.

Gómez, S., and C. Chesñevar. 2004. A hybrid approach to pattern classification using neural networks and defeasible argumentation. In: *Proc. 17th Intl. FLAIRS Conference*, 393–398. Miami, FL, American Assoc. for Artificial Intelligence.

Grosof, B. N., I. Horrocks, R. Volz, and S. Decker. 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., and R. Möller. 2001. RACER System Description. Technical Report, University of Hamburg, Computer Science Department.

Heymans, S., and D. Vermeir, 2002. A defeasible ontology language. In: *CoopIS/DOA/ODBASE*, 1033–1046.

Horridge, M., B. Parsia, and U. Sattler. 2008. Laconic and precise justifications in OWL. In: *Procs. of the VII International Semantic Web Conference (ISWC 2008)*, 323–338. Karlsruhe, Germany.

Horrocks, I. 1998. The FaCT System. In: *TABLEAUX '98: Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, 307–312. London, UK: Springer-Verlag.

Huang, Z., F. van Harmelen, and A. ten Teije. 2005. Reasoning with inconsistent ontologies. In: *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*, eds. L. P. Kaelbling and A. Saffiotti, 454–459. Scotland: Edinburgh.

Huang, Z., F. van Harmelen, A. ten Teije, P. Groot, and C. Visser. 2004. Reasoning with inconsistent ontologies: A general framework, Technical report, Department of Artificial Intelligence, Vrije Universiteit Amsterdam.

Kakas, A. C., P. Mancarella, and P. M. Dung. 1994. The acceptability semantics for logic programs. In: *Proceedings 11th. International Conference on Logic Programming*, 504–519. Santa Margherita, Italy: MIT Press.

Kakas, A. C., and F. Toni. 1999. Computing argumentation in logic programming. *Journal of Logic and Computation* 9(4):515–562.

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

Krötzch, M., S. Rudolph, and P. Hitzler. 2007. Complexity of horn description logics. Technical Report, Institute AIFB, Universität Karlsruhe, Germany.

Laera, L., V. Tamma, J. Euzenat, T. Bench-Capon, and T. Payne. 2006. Reaching agreement over ontology alignments. In: *Proc. 5th International Semantic Web Conference (ISWC 2006)*, Athens, GA.

Lam, S. C., J. Z. Pan, D. Sleeman, and W. Vasconcelos. 2006. A fine-grained approach to resolving unsatisfiable ontologies. In: *Proc. 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, 428–434.

Lloyd, J. 1987. *Foundations of Logic Programming*. Berlin, Germany: Springer-Verlag.

McGuiness, D. L., and F. van Harmelen. 2004. OWL Web Ontology Language Overview.

Meyer, T., K. Lee, R. Booth, and J. Z. Pan. 2006. Finding maximally satisfiable terminologies for the description logic $\mathcal{ALC}$. In: *Proceedings of AAAI-06*. Boston, MA.

Mitra, P. 2004. An algebraic framework for the interoperation of ontologies. PhD thesis, Dept. of Electrical Engineering Stanford University.

Nute, D. 1988. Defeasible reasoning. In: *Aspects of Artificial Intelligence*, ed. J. H. Fetzer, 251–288. Norwell, MA: Kluwer Academic Publishers.

Nute, D. 1992. Basic defeasible logic. In: *Intensional Logics for Programming*, ed. L. Fariñas del Cerro. Oxford: Claredon Press.

Parsia, B., and E. Sirin. 2004. Pellet: An OWL DL Reasoner. In: *3rd International Semantic Web Conference (ISWC2004)*. Hiroshima, Japan.

Parsons, S., C. Sierrra, and N. Jennings. 1998. Agents that reason and negotiate by arguing. *Journal of Logic and Computation* 8:261–292.

Pollock, J. L. 1974. *Knowledge and Justification*. Princeton, NJ: Princeton University Press.

Pollock, J. L. 1987. Defeasible reasoning. *Cognitive Science* 11:481–518.

Pollock, J. L. 1995. *Cognitive Carpentry: A Blueprint for How to Build a Person*. Boston: Bradford/MIT Press.

Prakken, H., and G. Sartor. 2002. The role of logic in computational models of legal argument – A critical survey. In: *Computational Logic: Logic Programming and Beyond*, eds. A. Kakas and F. Sadri, 342–380. Springer.

Prakken, H., and G. Vreeswijk. 2002. Logics for defeasible argumentation. In: *Handbook of Philosophical Logic*, eds. D. Gabbay and F. Guenthner, 219–318. London: Kluwer Academic Publisher.

Rahwan, I., S. D. Ramchurn, N. R. Jennings, P. Mcburney, S. Parsons, and L. Sonenberg. 2003. Argumentation-based negotiation. *Knowl. Eng. Rev.* 18(4):343–375.

Reiter, R., and G. Criscuolo. 1981. On interacting defaults. In: *Proceedings of the Seventh International Joint Conference on Artificial Intelligence (IJCAI'81)*, 94–100.

Ricca, F., and N. Leone. 2007. Disjunctive logic programming with types and objects: The DLV$^+$ system. *J. Applied Logic* 5(3):545–573.

Sandewall, E. 1986. Non-monotonic inference rules for multiple inheritance with exceptions. In: *Proceedings IEEE 74*, 1345–1353.

Sierra, C., and P. Noriega. 2002. Agent-mediated interaction. from auctions to negotiation and argumentation. In: *Foundations and Applications of Multi-Agent Systems – In LNCS Series*, Vol. 2403, 27–48. Springer.

Simari, G. R., and R. P. Loui. 1992. A mathematical treatment of defeasible reasoning and its implementation. *Artificial Intelligence* 53:125–157.

Stolzenburg, F., A. García, C. Chesñevar, and G. Simari. 2003. Computing generalized specificity. *J. N. Classical Logics* 13(1):87–113.

Terracina, G., E. D. Francesco, C. Panetta, and N. Leone. 2008. Enhancing a DLP system for advanced database applications. In: *LNCS*, eds. D. Calvanese and G. Lausen, RR, Vol. 5341, 119–134. Springer.

Verheij, B. 2005. *Virtual Arguments, On the Design of Argument Assistants for Lawyers and Other Arguers*. The Hague: Asser Press.

Volz, R. 2004. Web ontology reasoning with logic databases. PhD thesis, Universität Fridericiana zu Karlsruhe. Karlsruhe, Germany.

Wang, H., M. Horridge, A. Rector, N. Drummond, and J. Seidenberg. 2005. Debugging owl-dl ontologies: A heuristic approach. In: *ISWC 2005, LNCS 3729*, 745–757. Springer.

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

Zhang, P., J. Sun, and H. Chen. 2005. Frame-based argumentation for group decision task generation and identification. *Decision Support Systems* 39:643–659.

NOTES

1. http://www.w3c.org/
2. This example, as well as Example 2.3, are based on García and Simari (2004).
3. This example appears in García and Simari (2004) in a different context.
4. See `http://lidia.cs.uns.edu.ar/DeLP`
5. For example, $Convert(\{a(X) \leftarrow b(X)\}) = \{a(X) \prec\!\!\!- b(X)\}$.