Abstract. The Semantic Web is a vision of the current Web where resources have exact meaning assigned in terms of ontologies, thus enabling agents to reason about them. As inconsistencies cannot be treated by standard reasoning approaches, we use Defeasible Logic Programming (DeLP) to reason with possibly inconsistent ontologies. In this article we show how to integrate rules and ontologies in the Semantic Web. We show how to use a possibly inconsistent set of rules represented by a DeLP program to reason on top of a set of (possibly inconsistent) ontologies.

1 Introduction

The Semantic Web [1] (SW) is a vision of the current Web where resources have exact meaning assigned in terms of ontologies [2], thus enabling agents to reason about them. The Ontology Layer of the SW is well developed with the OWL language whose underlying semantics is based on the so-called Description Logics (DL) [3], for which specialized reasoners exist [4].

However, despite existing advances in SW-related technologies, there are still open research issues. In particular, the Rule Layer’s goal is to complement ontologies when ontology languages cannot fulfill all of the expressivity requirements for describing a domain, and its development is important as its semantics has not been standardized.

Besides, although existing reasoners provide efficient implementations for detecting inconsistent ontologies, they are incapable of performing inferences upon them. In a previous work [5], we presented a formalism called $\delta$-ontologies capable of reasoning with potentially inconsistent DL ontologies.

In this article, we extend the $\delta$-ontologies framework to suitably provide rules on top of $\delta$-ontologies. Our extension allows to model incomplete and possibly inconsistent set of rules, thus suitably extending the SW Rule Layer. Rules in the proposed extension are to be interpreted as DeLP program with special primitives to access the knowledge represented in the ontologies of the ontology layer. In this way rules and ontologies are integrated as a single DeLP program
upon which queries can be posed. The result of such queries will depend on the content of the rules as well as the contents of the underlying ontologies.

The rest of this paper is structured as follows. In Section 2 we briefly present the fundamentals of Description Logics and Defeasible Logic Programming. Section 3 briefly recalls the framework of $\delta$-ontologies for reasoning with possibly inconsistent ontologies. In Section 4, we extend the $\delta$-ontologies framework to allow for building rules on top of ontologies. Finally Section 5 discusses related work and Section 6 concludes the paper.

2 Background

2.1 Description Logics

Description Logics (DL) are a well-known family of knowledge representation formalisms [3]. They are based on the notions of concepts (unary predicates, classes) and roles (binary relations), and are mainly characterized by the constructors that allow complex concepts and roles to be built from atomic ones. Let $C$ and $D$ stand for concepts and $R$ for a role name. Concept descriptions are built from concept names using the constructors conjunction ($C \sqcap D$), disjunction ($C \sqcup D$), negation ($\neg C$), existencial restriction ($\exists R.C$), and value restriction ($\forall R.C$). To define the semantics of concept descriptions, concepts are interpreted as subsets of a domain of interest, and roles as binary relations over this domain. Further extensions to the basic DL are possible including inverse and transitive roles noted as $P^-$ and $P^+$, resp.

A DL ontology consists of two finite and mutually disjoint sets: a Tbox which introduces the terminology and an Abox which contains facts about particular objects in the application domain. Tbox statements have the form $C \sqsubseteq D$ (inclusions) and $C \equiv D$ (equalities), where $C$ and $D$ are (possibly complex) concept descriptions. Objects in the Abox are referred to by a finite number of individual names and these names may be used in two types of assertional statements: concept assertions of the type $a : C$ and role assertions of the type $\langle a, b \rangle : R$, where $C$ is a concept description, $R$ is a role name, and $a$ and $b$ are individual names.

2.2 Defeasible Logic Programming

Defeasible Logic Programming (DeLP) [6] provides a language for knowledge representation and reasoning that uses defeasible argumentation [7] to decide between contradictory conclusions through a dialectical analysis. Codifying knowledge by means of a DeLP program provides a good trade-off between expressivity and implementability for dealing with incomplete and potentially contradictory information. In a defeasible logic program $P = (\Pi, \Delta)$, a set $\Delta$ of defeasible rules $P \leftarrow Q_1, \ldots, Q_n$, and a set $\Pi$ of strict rules $P \leftarrow Q_1, \ldots, Q_n$ can be distinguished. An argument $\langle A, H \rangle$ is a minimal non-contradictory set of ground defeasible clauses $A$ of $\Delta$ that allows to derive a ground literal $H$ possibly using ground rules of $\Pi$. Since arguments may be in conflict (concept captured in
terms of a logical contradiction), an attack relationship between arguments can be defined. A criterion is usually defined to decide between two conflicting arguments. If the attacking argument is strictly preferred over the attacked one, then it is called a proper defeater. If no comparison is possible, or both arguments are equi-preferred, the attacking argument is called a blocking defeater. In order to determine whether a given argument \( \mathcal{A} \) is ultimately undefeated (or warranted), a dialectical process is recursively carried out, where defeaters for \( \mathcal{A} \), defeaters for these defeaters, and so on, are taken into account. Given a DeLP program \( \mathcal{P} \) and a query \( \mathcal{H} \), the final answer to \( \mathcal{H} \) w.r.t. \( \mathcal{P} \) takes such dialectical analysis into account. The answer to a query can be: yes, no, undecided, or unknown.

3 Reasoning with Inconsistent Ontologies in DeLP

In the presence of inconsistent ontologies, traditional DL reasoners (such as Racer [4]) 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. In a previous work [5], we showed how DeLP can be used for coping with inconsistencies in ontologies such that the task of dealing with them is automatically solved by the reasoning system. We recall some of the concepts for making the article more self-contained.

**Definition 1 (δ-Ontology).** Let \( \mathcal{C} \) be an \( \mathcal{L}_b \)-class, \( \mathcal{D} \) an \( \mathcal{L}_h \)-class, \( \mathcal{A}, \mathcal{B} \) \( \mathcal{L}_{hb} \)-classes, \( \mathcal{P}, \mathcal{Q} \) properties, \( a, b \) individuals. Let \( \mathcal{T} \) be a set of inclusion and equality sentences in \( \mathcal{L}_{DL} \) of the form \( \mathcal{C} \sqsubseteq \mathcal{D} \), \( \mathcal{A} \equiv \mathcal{B} \), \( \top \sqsubseteq \forall \mathcal{P}. \mathcal{D} \), \( \top \sqsubseteq \forall \mathcal{P}^+. \mathcal{D} \), \( \mathcal{P} \sqsubseteq \mathcal{Q} \), \( \mathcal{P} \equiv \mathcal{Q} \), \( \mathcal{P} \equiv \mathcal{Q}^+ \), or \( \mathcal{P}^+ \sqsubseteq \mathcal{P} \) such that \( \mathcal{T} \) can be partitioned into two disjoint sets \( \mathcal{T}_S \) and \( \mathcal{T}_D \). Let \( A \) be a set of assertions disjoint with \( \mathcal{T} \) of the form \( a : \mathcal{D} \) or \( \langle a, b \rangle : \mathcal{P} \). A δ-ontology \( \Sigma \) is a tuple \((\mathcal{T}_S, \mathcal{T}_D, A)\). The set \( \mathcal{T}_S \) is called the strict terminology (or Sbox), \( \mathcal{T}_D \) the defeasible terminology (or Dbox) and \( A \) the assertional box (or Abox).

**Example 1.** Consider the δ-ontologies \( \Sigma_1 = (\emptyset, \mathcal{T}_D^1, A^1) \) about swimming and \( \Sigma_2 = (\mathcal{T}_S^2, \mathcal{T}_D^2, A^2) \) about programming both presented in Fig. 1. The defeasible terminology \( \mathcal{T}_D^1 \) says that both free and scuba divers are divers; saturation divers are scuba divers; somebody who swims a race stroke is usually a race swimmer, and someone who can swim a rescue stroke is normally considered a rescue swimmer. The assertional box \( A^1 \) establishes that crawl is a race stroke; side is a rescue stroke; John is able to swim both crawl and side strokes, and finally Paul is a saturation diver. The strict terminology \( \mathcal{T}_S^2 \) expresses that among programming languages, both logic programming and object-oriented languages can be found. The Dbox \( \mathcal{T}_D^2 \) says that a programmer is usually somebody who can program in some programming language unless he has failed the elementary programming course. The Abox \( A^2 \) establishes that Prolog is a logic programming language and that John can program in the Prolog programming language; that Java is an object-oriented language and that Mary can program Java code, and that Paul is capable of programming in the Java programming language although he failed the elementary programming course.
Swimming ontology $\Sigma_1 = (\emptyset, T_1^\text{D}, A^1)$:

- **Defeasible terminology** $T_1^\text{D}$:
  - $\text{Free\ Diver} \sqsubseteq \text{Diver}$; $\text{Saturation\ Diver} \sqsubseteq \text{Scuba\ Diver}$
  - $\exists \text{swims} \text{Race\ Stroke} \sqsubseteq \text{Race\ Swimmer}$; $\exists \text{swims} \text{Rescue\ Stroke} \sqsubseteq \text{Rescue\ Swimmer}$

- **Assertional box** $A^1$:
  - $\text{CRAWL} : \text{Race\ Stroke}$; $\text{SIDE} : \text{Rescue\ Stroke}$
  - $\langle \text{JOHN}, \text{CRAWL} \rangle : \exists \text{swims}$; $\langle \text{JOHN}, \text{SIDE} \rangle : \exists \text{swims}$; $\text{PAUL} : \text{Saturation\ Diver}$

Programming ontology $\Sigma_2 = (T_2^\text{S}, T_2^\text{D}, A^2)$:

- **Strict terminology** $T_2^\text{S}$:
  - $\text{LP\ Lang} \sqsubseteq \text{OOP\ Lang} \sqsubseteq \text{Lang}$

- **Defeasible terminology** $T_2^\text{D}$:
  - $\exists \text{programs} \text{Lang} \sqsubseteq \text{Programmer}$
  - $\exists \text{programs} \text{Lang} \sqcap \text{Failed\Prog\_101} \sqsubseteq \neg \text{Programmer}$

- **Assertional box** $A^2$:
  - $\text{PROLOG} : \text{LP\ Lang}$; $\text{JAVA} : \text{OOP\ Lang}$
  - $\langle \text{JOHN}, \text{PROLOG} \rangle : \exists \text{programs}$; $\langle \text{MARY, JAVA} \rangle : \exists \text{programs}$
  - $\langle \text{PAUL, JAVA} \rangle : \exists \text{programs}$

For assigning semantics to a $\delta$-ontology we defined two translation functions $T_\Delta$ and $T_{\Pi}$ from DL to DeLP based on the work of [8]. We recall some of the definitions, for details see [5].

**Definition 2.** ($T_\Pi$ mapping from DL sentences to DeLP strict rules) Let $A, C, D$ be concepts, $X, Y$ variables, $P, Q$ properties. The $T_\Pi^* : \mathcal{L}^{DL} \rightarrow \mathcal{L}^{DeLP}$ mapping is defined in Fig. 2. Besides, intermediate transformations of the form $(H_1 \land H_2) \leftarrow B$ will be rewritten as two rules “$H_1 \leftarrow B$” and “$H_2 \leftarrow B$”.

Similarly transformations of the form “$H_1 \leftarrow B \land H_2$”, and rules of the form “$H \leftarrow (B_1 \lor B_2)$” will be rewritten as two rules “$H \leftarrow B_1$” and “$H \leftarrow B_2$”.

**Definition 3 (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 “$\text{Trans}(r)$”, is defined as:

$$\text{Trans}(r) = \{ H \leftarrow B_1, B_2, \ldots, B_{n-1}, B_n \}.$$

**Definition 4 ($T_{\Pi}$ mapping from DL sentences to DeLP strict rules).**

We define the mapping from DL ontologies into DeLP strict rules as $T_{\Pi}(T) = \text{Trans}(T_{\Pi}(T))$. 

Fig. 1. Ontologies $\Sigma_1$ and $\Sigma_2$
\[ T_H((C \subseteq D)) =_{df} \{ T_h(D, X) \leftarrow T_h(C, X) \} , \]
if C is an $\aleph_k$-class and D an $\aleph_k$-class
\[ T_H((C = D)) =_{df} T_H((C \subseteq D)) \cup T_H((D \subseteq C)) ; \]
if C and D are $\aleph_k$-classes
\[ T_H((\top \subseteq \forall P.D)) =_{df} \{ T_h(D, Y) \leftarrow P(X, Y) \} , \]
if D is an $\aleph_k$-class
\[ T_H((\top \subseteq \forall P^- D)) =_{df} \{ T_h(D, X) \leftarrow P(X, Y) \} , \]
if D is an $\aleph_k$-class
\[ T_H((a : D)) =_{df} \{ T_h(D, a) \} , \]
if D is an $\aleph_k$-class
where:
\[ T_h(A, X) =_{df} A(X) \]
\[ T_h((C \cap D), X) =_{df} T_h(C, X) \land T_h(D, X) \]
\[ T_h((\forall R.C), X) =_{df} T_h(C, Y) \leftarrow R(X, Y) \]
\[ T_h(A, X) =_{df} A(X) \]
\[ T_h((C \cap D), X) =_{df} T_h(C, X) \land T_h(D, X) \]
\[ T_h((C \cup D), X) =_{df} T_h(C, X) \lor T_h(D, X) \]
\[ T_h((\exists R.C), X) =_{df} R(X, Y) \land T_h(C, Y) \]

**Fig. 2.** Mapping from DL ontologies to DeLP strict rules

**Definition 5 (Interpretation of a $\delta$-ontology).** Let $\Sigma = (T_S, T_D, A)$ be a $\delta$-ontology. The interpretation of $\Sigma$ is a DeLP program $P = (T_H(T_S) \cup T_H(A), T_D(T_D))$.

Notice that in order to keep consistency within an argument, we must enforce some internal coherence between the Abox and the Tbox; namely given a $\delta$-ontology $\Sigma = (T_S, T_D, A)$, it must not be possible to derive two complementary literals from $T_H(T_S) \cup T_H(A)$.

**Definition 6. (Potential, justified and strict membership of an individual to a class)** Let $\Sigma = (T_S, T_D, A)$ be a $\delta$-ontology, C a class name, a an individual, and $P = (T_H(T_S) \cup T_H(A), T_D(T_D))$.

1. The individual a potentially belongs to class C, noted as $\text{PotentialMember}(a, C, \Sigma)$, iff there exists an argument $\langle A, C(a) \rangle$ w.r.t. $P$;
2. the individual a justifiedly belongs to class C, noted as $\text{JustifiedMember}(a, C, \Sigma)$, iff there exists a warranted argument $\langle A, C(a) \rangle$ w.r.t. $P$; and,
3. the individual a strictly belongs to class C, noted as $\text{StrictMember}(a, C, \Sigma)$, iff there exists an argument $\langle 0, C(a) \rangle$ w.r.t. $P$.

**Example 2 (Continues Ex. 1).** Consider again the $\delta$-ontologies $\Sigma_1$ and $\Sigma_2$, they are interpreted as the DeLP programs $P_1$ and $P_2$ according to Def. 5 as shown in Fig. 3. From $P_1$, we can determine that John justifiedly belongs to the concept Race$\_Swimmer$ in $\Sigma_1$ as there exists a warranted argument structure
\( \langle A_1, \text{race_swimmer}(john) \rangle \) where:

\[ A_1 = \{ \text{race_swimmer}(john) \leftarrow \text{swims}(john, \text{crawl}), \text{race_stroke}(\text{crawl}) \} \]

Likewise, there are warranting arguments \( A_2 \) and \( A_3 \) for \( \text{rescue_swimmer}(john) \) and \( \text{diver}(paul) \) resp., with:

\[ A_2 = \{ \text{rescue_swimmer}(john) \leftarrow \text{swims}(john, \text{side}), \text{race_stroke}(\text{side}) \} \]
\[ A_3 = \{ \text{diver}(paul) \leftarrow \text{scuba_diver}(paul) \\
\text{scuba_diver}(paul) \leftarrow \text{saturation_diver}(paul) \} \cdot \]

From \( P_2 \) in turn we can conclude that both John and Mary justifiedly belong to the concept Programmer but Paul justifiedly belongs to the concept \( \sim \text{Programmer} \) as there are warranted arguments \( \langle B_1, \text{programmer}(john) \rangle, \langle B_2, \text{programmer}(mary) \rangle \), and \( \langle B_3, \sim \text{programmer}(paul) \rangle \), where:

\[ B_1 = \{ \text{programmer}(john) \leftarrow \text{programs}(john, \text{prolog}) \} \cdot \]
\[ B_2 = \{ \text{programmer}(mary) \leftarrow \text{programs}(mary, \text{java}) \} \cdot \]
\[ B_3 = \{ \sim \text{programmer}(paul) \leftarrow \text{programs}(paul, \text{java}), \text{failed_prog_{101}}(paul) \} \cdot \]

Notice that there exists another argument \( \langle B_4, \text{programmer}(paul) \rangle \) with \( B_4 = \{ \text{programmer}(paul) \leftarrow \text{programs}(paul, \text{java}) \} \) that is defeated by argument \( B_3 \).

---

**DeLP program** \( P_1 = (\Pi_1, \Delta_1) \) obtained from \( \Sigma_1 \):

**Facts** \( \Pi_1 \):
- race_stroke(crawl).
- race_stroke(side).
- saturation_diver(paul).
- swims(john, crawl).
- swims(john, side).

**Defeasible rules** \( \Delta_1 \):
- diver(X) \leftarrow free_diver(X).
- diver(X) \leftarrow scuba_diver(X).
- scuba_diver(X) \leftarrow saturation_diver(X).
- race_swimmer(X) \leftarrow swims(X, Y), race_stroke(Y).
- race_swimmer(X) \leftarrow swims(X, Y), race_stroke(Y).

**DeLP program** \( P_2 = (\Pi_2, \Delta_2) \) obtained from \( \Sigma_2 \):

**Facts and strict rules** \( \Pi_2 \):
- lp_lang(prolog).
- oop_lang(java).
- programs(john, prolog).
- programs(mary, java).
- programs(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** \( \Delta_2 \):
- programmer(X) \leftarrow programs(X, Y), lang(Y).
- \sim programmer(X) \leftarrow programs(X, Y), lang(Y), failed_prog_{101}(X).

**Fig. 3.** DeLP programs \( P_1 \) and \( P_2 \) obtained from ontologies \( \Sigma_1 \) and \( \Sigma_2 \), resp.
4 Adding Rules on Top of Ontologies

We now define how to express rules in the Semantic Web in the presence of incompleteness and potential inconsistency. The notions presented will lead to the central definition of integration system that joins rules and ontologies making it suitable for a SW setting.

Definition 7 (Strict, justified and potential membership statements). Let $a$ be an individual name, $C$ a concept name, and $\Sigma$ a $\delta$-ontology. The expression “$\text{StrictMember}(a, C, \Sigma)$” is called a strict membership statement and queries if “$a$” strictly belongs to “$C$” w.r.t. $\Sigma$. The expression “$\text{JustifiedMember}(a, C, \Sigma)$” is called a justified membership statement and queries if “$a$” justifiably belongs to “$C$” w.r.t. $\Sigma$. The expression “$\text{PotentialMember}(a, C, \Sigma)$” is called a potential membership statement and queries if “$a$” potentially belongs to “$C$” w.r.t. $\Sigma$.

Definition 8 (Semantic web strict rule). A semantic web strict rule is an ordered pair, denoted “$B \implies H$”, whose first member, $B$, is a finite set of literals or potential membership statements, and whose second member, $H$, is a literal. A semantic web strict rule with antecedent $\{L_1, \ldots, L_n\}$ and head $H$ will be also written as “$L_1 \land \ldots \land L_n \implies H$.”

Definition 9 (Semantic web defeasible rule). A semantic web defeasible rule is an ordered pair, denoted “$B \sqsupseteq H$”, whose first member, $B$, is a finite set of literals or potential membership statements, and whose second member, $H$, is a literal. A semantic web strict rule with antecedent $\{L_1, \ldots, L_n\}$ and head $H$ will be also written as “$L_1 \land \ldots \land L_n \sqsupseteq H$.”

Definition 10 (Semantic web program). Let $S$ be a set of semantic web strict rules and $D$ a set of semantic web defeasible rules. A semantic web program is a pair $\langle S, D \rangle$.

Definition 11 (Integration system). Let $\mathcal{P}$ be a semantic web program and let $\Sigma_1, \ldots, \Sigma_n$ be $n$ $\delta$-ontologies. An integration system of rules and ontologies $I$ is a pair $\langle \mathcal{P}, \{\Sigma_i\}_{i=1,\ldots,n} \rangle$.

Example 3. Consider the semantic web program $\mathcal{P} = \langle S, D \rangle$ presented in Fig. 4, this SW program will be integrated with ontologies $\Sigma_1$ and $\Sigma_2$ from Ex. 1 into the integration system $I = \langle \mathcal{P}, \{\Sigma_1, \Sigma_2\} \rangle$. In $\mathcal{P}$, the set of strict semantic web rules $S$ expresses that somebody who potentially belongs to the concept “race swimmer” (resp. “rescue swimmer”) in ontology $\Sigma_1$ is a race swimmer (resp. rescue swimmer) and that whoever is a potential member of the concept “programmer” in ontology $\Sigma_2$ is a computer geek. The set of defeasible semantic web rules $D$ says that computer geeks are not usually good at sports but expert swimmers normally are; if somebody is either capable of swimming both a race stroke and a rescue stroke the he is often considered an expert swimmer; finally, a diver is usually considered an expert swimmer.
Set of strict semantic web rules $S$:

$\forall \text{PotentialMember}(x, \text{Race Swimmer}, \Sigma_1) \implies \text{Race Swimmer}(x)$
$\forall \text{PotentialMember}(x, \text{Rescue Swimmer}, \Sigma_2) \implies \text{Rescue Swimmer}(x)$
$\forall \text{PotentialMember}(x, \text{Programmer}, \Sigma_2) \implies \text{Geek}(x)$

Set of defeasible semantic web rules $D$:

$\text{Geek}(x) \gg \neg \text{Good}(x)$
$\text{Swimmer}(x) \gg \text{Good}(x)$
$\text{Race Swimmer}(x) \land \text{Rescue Swimmer}(x) \gg \text{Swimmer}(x)$
$\forall \text{PotentialMember}(x, \text{Diver}, \Sigma_1) \gg \text{Swimmer}(x)$

Fig. 4. Semantic web program $P = \langle S, D \rangle$

In order to answer queries posed against an integration system of rules and ontologies, we will interpret integration systems as DeLP programs. We define next the notions of semantic interpretation and answer to a query for an integration system.

**Definition 12 (Semantic interpretation).** Let $I = (P, \Sigma_1, \ldots, \Sigma_n)$ be an integration system such that: $P = (H^P, \Delta^P)$, $\Sigma_1 = (T_S^1, T_D^1, A^1)$, $\ldots$, $\Sigma_n = (T_S^n, T_D^n, A^n)$. The semantic interpretation of $I$, noted as $\text{Sem}(I)$, is the DeLP program:

$\phi(H^P) \cup \bigcup_{i=1,\ldots,n} T(T_S^i) \cup \bigcup_{i=1,\ldots,n} T(A^i), \phi(\Delta^P) \cup \bigcup_{i=1,\ldots,n} T(T_D^i))$.

**Definition 13 (Answer to a query in a SW integration system).** Let $I$ be a SW integration system and $L$ a literal. The answer to the query $L$, noted as $\text{Answer}_I(L)$, is defined as:

- **Yes** iff the answer to the query $L$ is Yes w.r.t. $\text{Sem}(I)$;
- **No** iff the answer to the query $\neg L$ is Yes w.r.t. $\text{Sem}(I)$, and
- **Undecided** iff the answer to the query $L$ is Undecided. w.r.t. $\text{Sem}(I)$.

**Example 4 (Continues Ex. 3).** Consider again the integration system $I$ presented in Ex. 3. When we compute $\text{Sem}(I)$, we obtain the DeLP program formed by the fragments presented in Fig. 5 along with the ones already presented in Ex. 2. We will show that the answer for the query $\text{good}(\text{john})$ is **Undecided**, for $\text{good}(\text{mary})$ is **No** and for $\text{good}(\text{paul})$ is **Yes**.

First, we will consider the dialectical analysis for the query “$\text{good}(\text{john})$”. There exists an argument $\langle C_1, \text{good}(\text{john}) \rangle$ where:

$C_1 = A_1 \cup A_2 \cup$

$\{ (\text{good}(\text{john}) \prec \text{swimmer}(\text{john})),$
$\quad (\text{swimmer}(\text{john}) \prec \text{race_swimmer}(\text{john}), \text{rescue_swimmer}(\text{john})) \}$.

However, there is an argument $\langle C_2, \neg \text{good}(\text{john}) \rangle$, that says John is not good at sports as he is a geek because he is a programmer, that defeats argument $C_1$, where $C_2 = B_1 \cup \{ \neg \text{good}(\text{john}) \prec \text{geek}(\text{john}) \}$. Therefore, the answer for the query “$\text{good}(\text{john})$” is **Undecided**.
When we consider the dialectical analysis for determining the answer to the query “good(mary)”, we find out that there is a warranted argument \( \langle B_2 \cup \{ \text{geek}(mary) \rightarrow \text{programmer}(mary) \rangle, \sim \text{good}(mary) \rangle \).

Last, let us consider the dialectical tree for the literal “good(paul)”. There is an argument \( \langle D_1, \text{good}(paul) \rangle \), based on the defeasible information that asserts that Paul is an expert swimmer (because he is a saturation diver), with:

\[
D_1 = A_3 \cup \left\{ \langle \text{good}(paul) \rightarrow \text{swimmer}(paul) \rangle, \langle \text{swimmer}(paul) \rightarrow \text{diver}_{\Sigma_2}(paul) \rangle \right\}.
\]

But argument \( D_1 \) is attacked by an argument \( \langle D_2, \sim \text{good}(paul) \rangle \), where:

\[
D_2 = B_4 \cup \left\{ \langle \sim \text{good}(paul) \rightarrow \text{geek}(paul) \rangle, \langle \text{geek}(paul) \rightarrow \text{programmer}(paul) \rangle \right\}.
\]

Nevertheless, as Paul failed the elementary programming course, this argument is defeated by argument \( B_3 \) (see Ex. 2), thus reinstating argument \( D_1 \).

\[
\Phi(B_1 \land \ldots \land B_n \Rightarrow A) =_{df} \Phi(A) \leftarrow \Phi(B_1) \ldots \Phi(B_1)
\]

\[
\Phi(B_1 \land \ldots \land B_n \gtrless A) =_{df} \Phi(A) \leftarrow \Phi(B_1) \ldots \Phi(B_1)
\]

\[
\Phi(L(x_1, \ldots , x_n)) =_{df} L(X_1, \ldots , X_n)
\]

\[
\Phi(\sim L(x_1, \ldots , x_n)) =_{df} \sim L(X_1, \ldots , X_n)
\]

\[
\Phi(\text{PotentialMember}(a, C, \Sigma)) =_{df} C_{\Sigma}(a)
\]

\[
\Phi(B_1 \land \ldots \land B_n \Rightarrow A) =_{df} \Phi(A) \leftarrow \Phi(B_1) \ldots \Phi(B_1)
\]

\[
\Phi(B_1 \land \ldots \land B_n \gtrless A) =_{df} \Phi(A) \leftarrow \Phi(B_1) \ldots \Phi(B_1)
\]

\[
\Phi(L(x_1, \ldots , x_n)) =_{df} L(X_1, \ldots , X_n)
\]

\[
\Phi(\sim L(x_1, \ldots , x_n)) =_{df} \sim L(X_1, \ldots , X_n)
\]

\[
\Phi(\text{PotentialMember}(a, C, \Sigma)) =_{df} C_{\Sigma}(a)
\]

\[
\Phi(B_1 \land \ldots \land B_n \Rightarrow A) =_{df} \Phi(A) \leftarrow \Phi(B_1) \ldots \Phi(B_1)
\]

\[
\Phi(B_1 \land \ldots \land B_n \gtrless A) =_{df} \Phi(A) \leftarrow \Phi(B_1) \ldots \Phi(B_1)
\]

\[
\Phi(L(x_1, \ldots , x_n)) =_{df} L(X_1, \ldots , X_n)
\]

\[
\Phi(\sim L(x_1, \ldots , x_n)) =_{df} \sim L(X_1, \ldots , X_n)
\]

\[
\Phi(\text{PotentialMember}(a, C, \Sigma)) =_{df} C_{\Sigma}(a)
\]

\[
\Phi(B_1 \land \ldots \land B_n \Rightarrow A) =_{df} \Phi(A) \leftarrow \Phi(B_1) \ldots \Phi(B_1)
\]

\[
\Phi(B_1 \land \ldots \land B_n \gtrless A) =_{df} \Phi(A) \leftarrow \Phi(B_1) \ldots \Phi(B_1)
\]

\[
\Phi(L(x_1, \ldots , x_n)) =_{df} L(X_1, \ldots , X_n)
\]

\[
\Phi(\sim L(x_1, \ldots , x_n)) =_{df} \sim L(X_1, \ldots , X_n)
\]

\[
\Phi(\text{PotentialMember}(a, C, \Sigma)) =_{df} C_{\Sigma}(a)
\]

\[
\Phi(B_1 \land \ldots \land B_n \Rightarrow A) =_{df} \Phi(A) \leftarrow \Phi(B_1) \ldots \Phi(B_1)
\]

\[
\Phi(B_1 \land \ldots \land B_n \gtrless A) =_{df} \Phi(A) \leftarrow \Phi(B_1) \ldots \Phi(B_1)
\]

\[
\Phi(L(x_1, \ldots , x_n)) =_{df} L(X_1, \ldots , X_n)
\]

\[
\Phi(\sim L(x_1, \ldots , x_n)) =_{df} \sim L(X_1, \ldots , X_n)
\]

\[
\Phi(\text{PotentialMember}(a, C, \Sigma)) =_{df} C_{\Sigma}(a)
\]

Fig. 5. Interpretation of Semantic Web rules as DeLP rules

\[
\Phi(B_1 \land \ldots \land B_n \Rightarrow A) =_{df} \Phi(A) \leftarrow \Phi(B_1) \ldots \Phi(B_1)
\]

\[
\Phi(B_1 \land \ldots \land B_n \gtrless A) =_{df} \Phi(A) \leftarrow \Phi(B_1) \ldots \Phi(B_1)
\]

\[
\Phi(L(x_1, \ldots , x_n)) =_{df} L(X_1, \ldots , X_n)
\]

\[
\Phi(\sim L(x_1, \ldots , x_n)) =_{df} \sim L(X_1, \ldots , X_n)
\]

\[
\Phi(\text{PotentialMember}(a, C, \Sigma)) =_{df} C_{\Sigma}(a)
\]

Fig. 6. DeLP program \( P' = (\Pi_P, \Delta_P) \) obtained from the interpretation of \( P = (S, D) \)

5 Related Work

Eiter et al. [9] propose a combination of logic programming under the answer set semantics with the DLs \( SHIF(D) \) and \( SHOIN(D) \). This combination allows for building rules on top of ontologies as we do. However, in contrast to our approach, they are not able to handle inconsistencies neither in the ontologies nor in the rule bases. Williams & Hunter [10] 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.
6 Conclusions

We have presented a novel approach for combining rules and ontologies in the Semantic Web. The proposed approach allows to add incomplete and possibly inconsistent rules on top of also possibly inconsistent ontologies by interpreting them as DeLP programs. We have presented a framework for characterizing the behavior of the proposed approach and an example scenario. In spite of the results we have obtained, we think that we have a lot of work ahead as the formal properties arising from the approach must be characterized. Other research issue is related to the inclusion of both strict and justified membership statements in Semantic Web rules as in this work we have only considered the inclusion of potential membership statements. Our current research efforts are directed toward solving these issues.

Acknowledgments: Esta línea de investigación está financiada por la Agencia Nacional de Promoción Científica y Tecnológica (PICT 2002 No. 13.096, PICT 2003 No. 15.043, PAV 2004 076), y por los Proyectos PIP 112-200801-02798 (CONICET, Argentina), TIN2006-15662-C02-01 (MEC, Spain), PGI 24/ZN10 (SGCyT, UNS, Argentina) y la Universidad Nacional del Sur.

References