Integration of Rules and Ontologies with Defeasible Logic Programming

Sergio A. Gómez, Carlos I. Chesñevar, and Guillermo R. Simari

Artificial Intelligence Research and Development Laboratory
Department of Computer Science and Engineering
Universidad Nacional del Sur
Av. Alem 1253, (8000) Bahía Blanca, Argentina
Email: {sag,cic,grs}@cs.uns.edu.ar

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 δ-ontologies capable of reasoning with potentially inconsistent DL ontologies.

In this article, we extend the δ-ontologies framework to suitably provide rules on top of δ-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 δ-ontologies for reasoning with possibly inconsistent ontologies. In Section 4, we extend the δ-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 $\mathcal{P} = (\Pi, \Delta)$, a set Δ of defeasible rules $P \prec Q_1, \ldots, Q_n$, and a set Π of strict rules $P \leftarrow Q_1, \ldots, Q_n$ can be distinguished. An *argument* $\langle \mathcal{A}, H\rangle$ is a minimal non-contradictory set of ground defeasible clauses $\mathcal{A}$ of Δ that allows to derive a ground literal H possibly using ground rules of Π. Since arguments may be in conflict (concept captured in

terms of a logical contradiction), an attack relationship between arguments can be defined. A criterion is usually defined to decide between two conflicting arguments. If the attacking argument is strictly preferred over the attacked one, then it is called a *proper defeater*. If no comparison is possible, or both arguments are equi-preferred, the attacking argument is called a *blocking defeater*. In order to determine whether a given argument $\mathcal{A}$ is ultimately undefeated (or *warranted*), a dialectical process is recursively carried out, where defeaters for $\mathcal{A}$, defeaters for these defeaters, and so on, are taken into account. Given a DeLP program $\mathcal{P}$ and a query H, the final answer to H w.r.t. $\mathcal{P}$ takes such dialectical analysis into account. The answer to a query can be: *yes*, *no*, *undecided*, or *unknown*.

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 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).*

Example 1. Consider the δ-ontologies $\Sigma_1 = (\emptyset, T_D^1, A^1)$ about swimming and $\Sigma_2 = (T_S^2, T_D^2, A^2)$ about programming both presented in Fig. 1. The defeasible terminology 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 T_S^2 expresses that among programming languages, both logic programming and object-oriented languages can be found. The Dbox T_D^2 says that a programmer is usually somebody who can program in some programming language unless she 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 $\Sigma_1 = (\emptyset, T_D^1, A^1)$**:**

Defeasible terminology T_D^1**:**
Free_Diver $\sqcup$ Scuba_Diver $\sqsubseteq$ Diver; Saturation_Diver $\sqsubseteq$ Scuba_Diver
$\exists$swims.Race_Stroke $\sqsubseteq$ Race_Swimmer; $\exists$swims.Rescue_Stroke $\sqsubseteq$ Rescue_Swimmer

Assertional box A^1**:**
CRAWL : Race_Stroke; SIDE : Rescue_Stroke
$\langle$JOHN, CRAWL$\rangle$: swims; $\langle$JOHN, SIDE$\rangle$: swims; PAUL : Saturation_Diver

Programming ontology $\Sigma_2 = (T_S^2, T_D^2, A^2)$**:**

Strict terminology T_S^2**:**
LP_Lang $\sqcup$ OOP_Lang $\sqsubseteq$ Lang

Defeasible terminology T_D^2**:**
$\exists$programs.Lang $\sqsubseteq$ Programmer; $\exists$programs.Lang $\sqcap$ Failed_Prog_101 $\sqsubseteq \neg$Programmer

Assertional box A^2**:**
PROLOG : LP_Lang; JAVA : OOP_Lang
$\langle$JOHN, PROLOG$\rangle$: programs; $\langle$MARY, JAVA$\rangle$: programs
$\langle$PAUL, JAVA$\rangle$: programs PAUL : Failed_Prog_101

Fig. 1. Ontologies Σ_1 and Σ_2

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

Definition 2. ($\mathcal{T}_\Pi^*$ 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 Fig. 2. Besides, intermediate 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 "$H \leftarrow B_2$".*

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 "$\mathfrak{Trans}(r)$", is defined as:*

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

Definition 4 ($\mathcal{T}_\Pi$ mapping from DL sentences to DeLP strict rules). *We define the mapping from DL ontologies into DeLP strict rules as $\mathcal{T}_\Pi(T) = \mathfrak{Trans}(\mathcal{T}_\Pi^*(T))$.*

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

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

Definition 5 (Interpretation of a δ-ontology). *Let $\Sigma = (T_S, T_D, A)$ be a δ-ontology. The* interpretation of Σ *is a DeLP program $\mathcal{P} = (\mathcal{T}_{\Pi}(T_S) \cup \mathcal{T}_{\Pi}(A), \mathcal{T}_{\Delta}(T_D))$.*

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 δ-ontology $\Sigma = (T_S, T_D, A)$, it must not be possible to derive two complementary literals from $\mathcal{T}_{\Pi}(T_S) \cup \mathcal{T}_{\Pi}(A)$.

Definition 6. (Potential, justified and strict membership of an individual to a class) *Let $\Sigma = (T_S, T_D, A)$ be a δ-ontology, C a class name, a an individual, and $\mathcal{P} = (\mathcal{T}_{\Pi}(T_S) \cup \mathcal{T}_{\Pi}(A), \mathcal{T}_{\Delta}(T_D))$.*

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

Example 2 (Continues Ex. 1). Consider again the δ-ontologies Σ_1 and Σ_2, they are interpreted as the DeLP programs $\mathcal{P}_1$ and $\mathcal{P}_2$ according to Def. 5 as shown in Fig. 3. From $\mathcal{P}_1$, we can determine that John justifiedly belongs to the concept Race_Swimmer in Σ_1 as there exists a warranted argument structure

$\langle \mathcal{A}_1, race\_swimmer(john) \rangle$ where:

$$\mathcal{A}_1 = \left\{ race\_swimmer(john) \prec swims(john, crawl), race\_stroke(crawl) \right\}.$$

Likewise, there are warranting arguments $\mathcal{A}_2$ and $\mathcal{A}_3$ for $rescue\_swimmer(john)$ and $diver(paul)$ resp., with:

$$\mathcal{A}_2 = \left\{ rescue\_swimmer(john) \prec swims(john, side), rescue\_stroke(side) \right\} \text{ and}$$
$$\mathcal{A}_3 = \left\{ \begin{array}{l} diver(paul) \prec scuba\_diver(paul) \\ scuba\_diver(paul) \prec saturation\_diver(paul) \end{array} \right\}.$$

From $\mathcal{P}_2$ in turn we can conclude that both John and Mary justifiedly belong to the concept Programmer but Paul justifiedly belongs to the concept ¬Programmer as there are warranted arguments $\langle \mathcal{B}_1, programmer(john) \rangle$, $\langle \mathcal{B}_2, programmer(mary) \rangle$, and $\langle \mathcal{B}_3, \sim programmer(paul) \rangle$, where:

$$\mathcal{B}_1 = \left\{ programmer(john) \prec programs(john, prolog) \right\},$$
$$\mathcal{B}_2 = \left\{ programmer(mary) \prec programs(mary, java) \right\}, \text{ and}$$
$$\mathcal{B}_3 = \left\{ \sim programmer(paul) \prec programs(paul, java), failed\_prog\_101(paul) \right\}.$$

Notice that there exists another argument $\langle \mathcal{B}_4, programmer(paul) \rangle$ with $\mathcal{B}_4 = \{programmer(paul) \prec programs(paul, java)\}$ that is defeated by argument $\mathcal{B}_3$.

DeLP program $\mathcal{P}_1 = (\Pi_1, \Delta_1)$ obtained from Σ_1:

Facts Π_1:
$race\_stroke(crawl)$. $rescue\_stroke(side)$. $saturation\_diver(paul)$.
$swims(john, crawl)$. $swims(john, side)$.

Defeasible rules Δ_1:
$diver(X) \prec free\_diver(X)$.
$diver(X) \prec scuba\_diver(X)$.
$scuba\_diver(X) \prec saturation\_diver(X)$.
$race\_swimmer(X) \prec swims(X, Y), race\_stroke(Y)$.
$rescue\_swimmer(X) \prec swims(X, Y), rescue\_stroke(Y)$.

DeLP program $\mathcal{P}_2 = (\Pi_2, \Delta_2)$ obtained from Σ_2:

Facts and strict rules Π_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 Δ_2:
$programmer(X) \prec programs(X, Y), lang(Y)$.
$\sim programmer(X) \prec programs(X, Y), lang(Y), failed\_prog\_101(X)$.

Fig. 3. DeLP programs $\mathcal{P}_1$ and $\mathcal{P}_2$ obtained from ontologies Σ_1 and Σ_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 Σ a δ-ontology. The expression "$\mathfrak{StrictMember}(a, C, \Sigma)$" is called a* strict membership statement *and queries if "a" strictly belongs to "C" w.r.t. Σ. The expression "$\mathfrak{JustifiedMember}(a, C, \Sigma)$" is called a* justified membership statement *and queries if "a" justifiedly belongs to "C" w.r.t. Σ. The expression "$\mathfrak{PotentialMember}(a, C, \Sigma)$" is called a* potential membership statement *and queries if "a" potentially belongs to "C" w.r.t. Σ.*

Definition 8 (Semantic web strict rule). *A* semantic web strict rule *is an ordered pair, denoted "$B \Longrightarrow H$", 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 "$L_1 \wedge \ldots \wedge L_n \Longrightarrow H$".*

Definition 9 (Semantic web defeasible rule). *A* semantic web defeasible rule *is an ordered pair, denoted "$B \succ\!\!= H$", 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 "$L_1 \wedge \ldots \wedge L_n \succ\!\!= H$".*

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

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

Example 3. Consider the semantic web program $\mathcal{P} = \langle \mathcal{S}, \mathcal{D} \rangle$ presented in Fig. 4, this SW program will be integrated with ontologies Σ_1 and Σ_2 from Ex. 1 into the integration system $\mathcal{I} = \langle \mathcal{P}, \{\Sigma_1, \Sigma_2\} \rangle$. In $\mathcal{P}$, the set of strict semantic web rules $\mathcal{S}$ expresses that somebody who potentially belongs to the concept "race swimmer" (resp. "rescue swimmer") in ontology Σ_1 is a race swimmer (resp. rescue swimmer) and that whoever is a potential member of the concept "programmer" in ontology Σ_2 is a computer geek. The set of defeasible semantic web rules $\mathcal{D}$ 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.

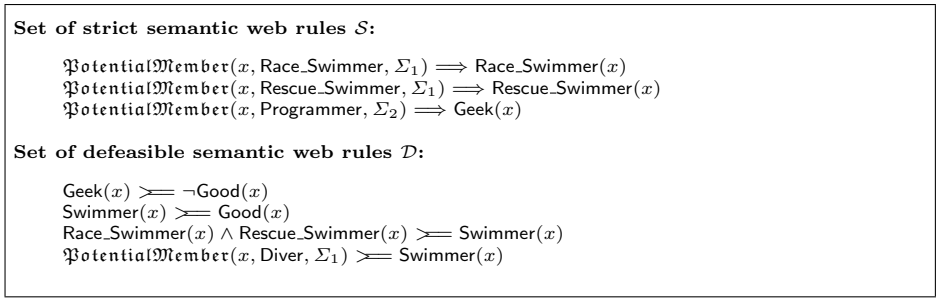

Fig. 4. Semantic web program $\mathcal{P} = \langle \mathcal{S}, \mathcal{D} \rangle$

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 $\mathcal{I} = (\mathcal{P}, \{\Sigma_1, \ldots, \Sigma_n\})$ be an integration system such that: $\mathcal{P} = \langle \Pi^{\mathcal{P}}, \Delta^{\mathcal{P}} \rangle$, $\Sigma_1 = (T_S^1, T_D^1, A^1)$, $\ldots$, $\Sigma_n = (T_S^n, T_D^n, A^n)$. The* semantic interpretation of $\mathcal{I}$*, noted as $\mathfrak{Sem}(\mathcal{I})$, is the DeLP program:*

$$\left(\Phi(\Pi^{\mathcal{P}}) \cup \bigcup_{i=1,\ldots,n} \mathcal{T}(T_S^i) \cup \bigcup_{i=1,\ldots,n} \mathcal{T}(A^i), \Phi(\Delta^{\mathcal{P}}) \cup \bigcup_{i=1,\ldots,n} \mathcal{T}(T_D^i)\right).$$

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

- YES *iff the answer to the query L is Yes w.r.t. $\mathfrak{Sem}(\mathcal{I})$;*
- NO *iff the answer to the query $\sim L$ is Yes w.r.t. $\mathfrak{Sem}(\mathcal{I})$, and*
- UNDECIDED *iff the answer to the query L is Undecided. w.r.t. $\mathfrak{Sem}(\mathcal{I})$.*

Example 4 (Continues Ex. 3). Consider again the integration system $\mathcal{I}$ presented in Ex. 3. When we compute $\mathfrak{Sem}(\mathcal{I})$, 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 $good(john)$ is UNDECIDED, for $good(mary)$ is NO and for $good(paul)$ is YES.

First, we will consider the dialectical analysis for the query "$good(john)$". There exists an argument $\langle \mathcal{C}_1, good(john) \rangle$ where:

$$\mathcal{C}_1 = \mathcal{A}_1 \cup \mathcal{A}_2 \cup \left\{ \begin{array}{l} (good(john) \prec swimmer(john)), \\ (swimmer(john) \prec race\_swimmer(john), rescue\_swimmer(john)) \end{array} \right\}.$$

However, there is an argument $\langle \mathcal{C}_2, \sim good(john) \rangle$, that says John is not good at sports as he is a geek because he is a programmer, that defeats argument $\mathcal{C}_1$, where $\mathcal{C}_2 = \mathcal{B}_1 \cup \{\sim good(john) \prec geek(john)\}$. Therefore, the answer for the query "$good(john)$" 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 $\langle \mathcal{B}_2 \cup \{geek(mary) \prec programmer(mary)\}, \sim good(mary)\rangle$.

Last, let us consider the dialectical tree for the literal "$good(paul)$". There is an argument $\langle \mathcal{D}_1, good(paul)\rangle$, based on the defeasible information that asserts that Paul is an expert swimmer (because he is a saturation diver), with:

$$\mathcal{D}_1 = \mathcal{A}_3 \cup \left\{ \begin{array}{l} (good(paul) \prec swimmer(paul)), \\ (swimmer(paul) \prec diver_{\Sigma_2}(paul)) \end{array} \right\}.$$

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

$$\mathcal{D}_2 = \mathcal{B}_4 \cup \left\{ \begin{array}{l} (\sim good(paul) \prec geek(paul)), \\ (geek(paul) \prec programmer(paul)) \end{array} \right\}.$$

Nevertheless, as Paul failed the elementary programming course, this argument is defeated by argument $\mathcal{B}_3$ (see Ex. 2), thus reinstating argument $\mathcal{D}_1$.

$$\begin{array}{rl} \Phi(B_1 \wedge \ldots \wedge B_n \Longrightarrow A) & =_{df} \Phi(A) \leftarrow \Phi(B_1), \ldots, \Phi(B_1) \\ \Phi(B_1 \wedge \ldots \wedge B_n \succ\!\!= A) & =_{df} \Phi(A) \prec \Phi(B_1), \ldots, \Phi(B_1) \\ \Phi(L(x_1, \ldots, x_n)) & =_{df} L(X_1, \ldots, X_n) \\ \Phi(\neg L(x_1, \ldots, x_n)) & =_{df} \sim L(X_1, \ldots, X_n) \\ \Phi(\mathfrak{PotentialMember}(a, C, \Sigma)) & =_{df} C_\Sigma(a) \end{array}$$

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

Strict rules $\Pi_\mathcal{D}$:

$race\_swimmer(X) \leftarrow race\_swimmer_{\Sigma_1}(X).$
$rescue\_swimmer(X) \leftarrow rescue\_swimmer_{\Sigma_1}(X).$
$geek(X) \leftarrow programmer_{\Sigma_2}(X).$

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

$\sim good(X) \prec geek(X).$
$good(X) \prec swimmer(X).$
$swimmer(X) \prec race\_swimmer(X), rescue\_swimmer(X).$
$swimmer(X) \prec diver_{\Sigma_1}(X).$

Fig. 6. DeLP program $\mathcal{P}' = (\Pi_\mathcal{S}, \Delta_\mathcal{D})$ obtained from the interpretation of $\mathcal{P} = \langle \mathcal{S}, \mathcal{D}\rangle$

5 Related Work

Eiter *et al.*[9] 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 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

1. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scient. American (2001)
2. Gruber, T.R.: A translation approach to portable ontologies. Knowledge Acquisition **5**(2) (1993) 199–220
3. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P., eds.: The Description Logic Handbook – Theory, Implementation and Applications. Cambridge University Press (2003)
4. Haarslev, V., Möller, R.: RACER System Description. Technical report, University of Hamburg, Computer Science Department (2001)
5. Gómez, S.A., Chesñevar, C.I., Simari, G.R.: An Argumentative Approach to Reasoning with Inconsistent Ontologies. In Meyer, T., Orgun, M.A., eds.: Proc. of the Knowledge Representation in Ontologies Workshop (KROW 2008). Volume CPRIT 90., Sydney, Australia (2008) 11–20
6. García, A., Simari, G.: Defeasible Logic Programming an Argumentative Approach. Theory and Prac. of Logic Program. **4**(1) (2004) 95–138
7. Chesñevar, C.I., Maguitman, A., Loui, R.: Logical Models of Argument. ACM Computing Surveys **32**(4) (December 2000) 337–383
8. Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description Logic Programs: Combining Logic Programs with Description Logics. WWW2003, May 20-24, Budapest, Hungary (2003)
9. Eiter, T., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining Answer Set Programming with Description Logics for the Semantic Web. KR 2004 (2004) 141–151
10. Williams, M., Hunter, A.: Harnessing ontologies for argument-based decision-making in breast cancer. Proc. of the Intl. Conf. on Tools with AI (ICTAI'07) (2007) 254–261