Some Classical Problems of Inheritance Networks in the Light of Defeasible Ontology Reasoning

Sergio Alejandro Gómez

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@cs.uns.edu.ar

Abstract. Reasoning with possibly inconsistent ontologies is an important reasearch topic for the implementation of the Semantic Web as they pose a problem for performing instance checking. We contend that Defeasible Logic Programming (DeLP) is a reliable tool for doing ontology reasoning when Description Logic ontologies can be interpreted as DeLP programs. In this work we present some classical problems of the field of inheritance networks and show how they are modeled as inconsistent ontologies and thus how the problem of instance checking is solved; we also show how issues in reasoning with argumentation frameworks based on Dung's grounded semantics are also solved when applied to ontology reasoning, and we revise the main algorithm for instance checking when using DeLP with inconsistent ontologies.

1 Introduction

Reasoning with Description Logics ontologies [1] is an important topic for the implementation of the Semantic Web [2]. In the Semantic Web, the meaning of data resources is defined in terms of ontologies thus making them machine processable. Traditional reasoners, such as Racer [3] or Pellet [4], infer a taxonomy of concepts from an ontology and are used, among other tasks, to determine the membership of individuals to concepts—a problem known as instance checking. Inconsistent ontologies pose a problem for this because when reasoners detect an inconsistency, in the best scenario, they point to the definition that is causing trouble. After this, the knowledge engineer has to debug the ontology, that is making it consistent again. Many times this is not possible for many reasons, one of them is that the field being modeled can be intrinsically contradictory, another is that the engineer may not have the authority to edit the ontology's contents. Many approaches have been proposed for dealing with this situation, such as paraconsistent logics [5] and belief revision [6] (for an in-depth revision of related work, see [7,8,9]).

The hypothesis of this work is that defeasible argumentation [10] is a reliable tool for reasoning with possibly inconsistent ontologies and, in this regard,

Defeasible Logic Programming, as an implementation of defeasible argumentation, is an appropriate tool for this. In [7], Gómez *et al.* developed a framework for ontology reasoning based on defeasible argumentation called *δ-ontologies*; reasoning on this program instead of on the ontology. Instance checking comprise performing a dialectical process weighing opposing reasons supporting and rejecting the membership of individuals to concepts.

The contribution of this paper is three-fold: (i) it consolidates research results on the δ-ontology framework presented previously by Gómez *et al.* [7,8] and Gómez and Simari [9], (ii) it discusses how some historical problems in inheritance networks [11] are solved in δ-ontologies and, finally, (iii) it shows how issues in reasoning with argumentation frameworks based on Dung's grounded semantics are also solved in the δ-ontologies framework. Our research methodology includes the gathering of representative examples in the literature and showing how the δ-ontologies framework handles the problem of instance checking in these examples. In brief, this contribution reaffirms the results of previous work and can be considered an expansion of past work in the field.

Outline: In Section 2 we present the fundamentals of Description Logic ontologies, Defeasible Logic Programming and how inconsistent ontologies are handled with DeLP. In Section 3 we present some classical problems of the field of inheritance networks and argumentation and how they are handled as δ-ontologies. Finally, in Section 4 we revise the main algorithm for computing instance checking, discuss future work and present the conclusions.

2 Fundamentals of Ontologies Treated in Defeasible Argumentation

2.1 Description Logic ontologies

Description Logics (DL) are a well-known family of knowledge representation formalisms [1]. They are based on the notions of *concepts* (unary predicates, classes) and *roles* (binary relations), and are mainly characterized by 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 ($\mathsf{C} \sqcap \mathsf{D}$), disjunction ($\mathsf{C} \sqcup \mathsf{D}$), negation ($\neg\mathsf{C}$), existential restriction ($\exists\mathsf{R}.\mathsf{C}$), and value restriction ($\forall\mathsf{R}.\mathsf{C}$). 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.

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. Inclusion Tbox statements have the form $\mathsf{C} \sqsubseteq \mathsf{D}$, where C and D are possibly complex concept descriptions, meaning that every individual of C is also a D. 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* $\mathsf{a} : \mathsf{C}$ (meaning the individual a is a member of concept C) and *role assertions* $\langle \mathsf{a}, \mathsf{b} \rangle : \mathsf{R}$ (meaning that a is related to b through the role R).

Many reasoning *Abox reasoning* tasks are defined in DL, in this work we are only insterested in *instance checking*, that refers to determining if an individual is a member of a certain class. An ontology is incoherent if it has empty concepts and inconsistent if it is contradictory.

2.2 Argumentation in Defeasible Logic Programming

Defeasible Logic Programming (DeLP) [12] provides a language for knowledge representation and reasoning that uses *defeasible argumentation* to decide between contradictory conclusions through a *dialectical analysis*, and providing a good trade-off between expressiveness and implementability for dealing with incomplete and potentially contradictory information. In a DeLP program $\mathcal{P} = (\Pi, \Delta)$, a set Π of strict rules $P \leftarrow Q_1, \ldots, Q_n$ (which encode certain knowledge), and a set Δ of defeasible rules $P \prec Q_1, \ldots, Q_n$ (which encode knowledge with possible exceptions) can be distinguished. As in logic programming, P is the head of the rule and $Q_1, \ldots, Q_n$ comprise its body. 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. To decide between two conflicting arguments, we will use *generalized specificity*—a syntactic criterion that prefers arguments more informed and arguments based on shorter derivations. 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*. 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. $\mathcal{P}$ is based on such dialectical analysis. The answer to a query can be: *Yes* (when there exists a warranted argument $\langle \mathcal{A}, H \rangle$), *No* (when there exists a warranted argument $\langle \mathcal{A}, \sim H \rangle$), *Undecided* (when neither $\langle \mathcal{A}, H \rangle$ nor $\langle \mathcal{A}, \sim H \rangle$ are warranted), or *Unknown* (when H does not belong to $\mathcal{P}$).

2.3 Expressing Description Logic Ontologies in DeLP with δ-Ontologies

In the presence of inconsistent ontologies, traditional DL reasoners issue an error message and the knowledge engineer must then debug the ontology (*i.e.* making it consistent) to be able to perform instance checking. Gómez *et al.* [7] 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. δ-ontologies comprise an important subset of DL ontologies that can be be translated into logic programming (DL $\mathcal{L}_b$-classes can be translated, using Lloyd-Topor's transformations, as bodies of DeLP rules, $\mathcal{L}_h$-classes as heads of

DeLP rules, and $\mathcal{L}_{hb}$-classes work both at bodies and heads of rules). An ontology will be separated into a strict terminology T_S that will be understood as a strict set of DeLP rules, a defeasible terminology T_D that will be translated as a set of defeasible rules and an assertional box A that will be considered a set of DeLP facts. Formally:

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 *roles,* a, b *individuals. Let* T *be a set of inclusion and equality DL axioms of the form* $\mathsf{C} \sqsubseteq \mathsf{D}$*,* $\mathsf{A} \equiv \mathsf{B}$*,* $\top \sqsubseteq \forall \mathsf{P}.\mathsf{D}$*,* $\top \sqsubseteq \forall \mathsf{P}^{-}.\mathsf{D}$*,* $\mathsf{P} \sqsubseteq \mathsf{Q}$*,* $\mathsf{P} \equiv \mathsf{Q}$*,* $\mathsf{P} \equiv \mathsf{Q}^{-}$*, or* $\mathsf{P}^{+} \sqsubseteq \mathsf{P}$ *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* $\mathsf{a} : \mathsf{D}$ *or* $\langle \mathsf{a}, \mathsf{b} \rangle : \mathsf{P}$*. 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).*

For giving semantics to a δ-ontology, two translation functions $\mathcal{T}_{\Delta}(\cdot)$ and $\mathcal{T}_{\Pi}(\cdot)$ from DL to DeLP are defined based on the work of [13] (for details see [7]). First, axioms are considered to be in negation-normal form, meaning that negations are pushed inward class expressions. Informally, an axiom of the form $\mathsf{C} \sqsubseteq \mathsf{D}$ will be translated as $d(X) \leftarrow c(X)$ when it belongs to the Sbox and as $d(X) \multimap c(X)$ when it is a member of the Dbox. Abox assertions of the form $\mathsf{a} : \mathsf{C}$ are translated as DeLP facts $c(a)$ and $\langle \mathsf{a}, \mathsf{b} \rangle : \mathsf{r}$ as $r(a, b)$. Moreover, a formula of the form $\exists \mathsf{r}.\mathsf{C} \sqsubseteq \mathsf{D}$ is translated as $d(X) \multimap r(X, Y), c(Y)$, and one of the form $\mathsf{C} \sqcup \mathsf{D} \sqsubseteq \mathsf{E}$ as two axioms $\mathsf{C} \sqsubseteq \mathsf{E}$ and $\mathsf{D} \sqsubseteq \mathsf{E}$. 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 not to lose possible inferences $\mathcal{T}_{\Pi}(T_S)$ computes *transposes* of rules (*e.g.*, $\mathcal{T}_{\Pi}(\{\mathsf{C} \sqsubseteq \mathsf{D}\})$ is $\{(d(X) \leftarrow c(X)), (\sim c(X) \leftarrow\sim d(X))\}$). To keep consistency within arguments obtained from a given Σ, it must not be possible to derive two complementary literals from $\mathcal{T}_{\Pi}(T_S) \cup \mathcal{T}_{\Pi}(A)$. *Instance checking* in δ-ontologies is redefined to handle possible inconsistencies while retaining classic DL functionality where there is none: If C is a class and a an individual, (i) a is a potential member of C iff there exists an argument $\langle \mathcal{A}, C(a) \rangle$ w.r.t. $\mathcal{P}$; (ii) a is a justified member of C iff there exists a warranted argument $\langle \mathcal{A}, C(a) \rangle$ w.r.t. $\mathcal{P}$; (iii) a is a strict member of C iff there exists an argument $\langle \emptyset, C(a) \rangle$ w.r.t. $\mathcal{P}$, and (iv) the membership of a to C is undecided iff there is no argument for $C(a)$.

3 Problems of Inheritance Networks in δ-Ontologies

Here we review classical examples in the light of its representation and reasoning within δ-ontologies. The examples presented in this section are based on [11] and [14], and are taken from the area of inheritance networks [15] and defeasible reasoning [16].

Example 1. Consider this δ-ontology $\Sigma_1 = (T_S, T_D, A)$ where:

$$T_S = \left\{ \begin{array}{l} \mathsf{Black\_widow} \sqsubseteq \mathsf{Spider} \\ \mathsf{Spider} \sqsubseteq \mathsf{Arachnid} \\ \mathsf{Pet} \sqsubseteq \mathsf{Lives\_with\_human} \\ \mathsf{Arachnid} \sqsubseteq \mathsf{Exoskeleton} \\ \mathsf{Mammal} \sqsubseteq \neg\mathsf{Exoskeleton} \\ \mathsf{Beagle} \sqsubseteq \mathsf{Dog} \\ \mathsf{Dog} \sqsubseteq \mathsf{Mammal} \end{array} \right\}, T_D = \left\{ \begin{array}{l} \mathsf{Dog} \sqsubseteq \mathsf{Pet} \\ \mathsf{Spider} \sqsubseteq \neg\mathsf{Pet} \end{array} \right\} \text{ and}$$

$$A = \left\{ \begin{array}{l} \mathsf{SPOT} : \mathsf{Beagle} \\ \mathsf{PEPA} : \mathsf{Black\_widow} \end{array} \right\}.$$

This ontology is interpreted as the DeLP program $\mathcal{P}_1 = (\Pi, \Delta)$ where:

$$\Pi = \left\{ \begin{array}{ll} spider(X) \leftarrow black\_widow(X). & \sim black\_widow(X) \leftarrow \sim spider(X). \\ arachnid(X) \leftarrow spider(X). & \sim spider(X) \leftarrow \sim arachnid(X). \\ lives\_with\_human(X) \leftarrow pet(X). & \sim pet(X) \leftarrow \sim lives\_with\_human(X). \\ exoskeleton(X) \leftarrow arachnid(X). & \sim arachnid(X) \leftarrow \sim exoskeleton(X). \\ \sim exoskeleton(X) \leftarrow mammal(X). & \sim mammal(X) \leftarrow exoskeleton(X). \\ dog(X) \leftarrow beagle(X). & \sim beagle(X) \leftarrow \sim dog(X). \\ mammal(X) \leftarrow dog(X). & \sim dog(X) \leftarrow \sim mammal(X). \\ beagle(spot). & black\_widow(pepa). \end{array} \right\}$$

$$\Delta = \{\ pet(X) \multimap dog(X).\ \sim pet(X) \multimap arachnid(X).\ \}$$

Bender points out that from this representation, we should be able to reach conclusions such as: "a black widow spider has an exoskeleton", "Spot is a dog", "beagles live with humans", "spiders are not pets" (not true because people in the real world keep tarantulas as pets), "dogs do not have exoskeletons", "pets are not arachnids", and "Spot is not a black widow spider".

Let us see what the possible conclusions of this DeLP program are: First, we see that the Pepa spider has an exoskeleton (in this case we an argument formed by strict rules—Pepa is a black widow spider, which is an arachnid, and arachnids have exoskeletons); so Pepa is a strict member of the Exoskeleton. Spot is a dog because he is a beagle (we have again an argument formed solely by strict rules). Spot lives with humans (Spot is a beagle, beagles are dogs, dogs are pets, pets live with humans). We reach one of the limits of the model when we query if Pepa lives with humans; in this case, the answer is undecided because there is no argument in favor of $lives\_with\_human(pepa)$. Pepa is not a pet (clearly, Pepa is a spider and spider are not pets). Spot has no exoskeleton (Spot is dog, dogs are mammals, mammals don't have exoskeleton). When we ask if pets are not arachnids, we note that we cannot build an argument (not even adding $pet(a)$ to Π). Again, asking if Spot is not a black widow spider, the answer that DeLP brings is undecided (as we have no argument for it). Clearly this limitations are motivated by two issues: the model is inherently incomplete and rule systems allow for less conclusions that first-order logic, which is good, otherwise an inconsistency would produce that anything is a conclusion of our system.

Adding additional relations {$\mathsf{Bird} \sqsubseteq \neg\mathsf{Milk}$; $\mathsf{Pigeon} \sqsubseteq \mathsf{Bird}$; $\mathsf{Pigeon} \sqsubseteq \mathsf{Milk}$; $\mathsf{Tweety} : \mathsf{Pigeon}$} to the previous example yields the additional rules {$\sim milk(X) \multimap bird(X)$; $bird(X) \leftarrow pigeon(X)$; $\sim pigeon(X) \leftarrow \sim bird(X)$; $milk(X) \multimap pigeon(X)$; $pigeon(tweety)$}. This example codifies the knowledge that we believe that mammals give milk and

birds do not. But, actually, pigeons secrete a cheeselike substance, called pigeon's milk, into their crops and regurgitate it for nestlings. This program contains a contradiction because we can conclude that pigeons do not milk (thus contradicting that pigeons milk). In this case we conclude that Tweety milks because he is a pigeon and pigeons milk (there is an argument $\langle \mathcal{A}, \sim milk(tweety) \rangle$ that is defeated by $\langle \mathcal{B}, milk(tweety) \rangle$), where: $\mathcal{A} = \{\sim milk(tweety) \multimap\!\prec bird(tweety)\}$ and $\mathcal{B} = \{milk(tweety) \multimap\!\prec pigeon(tweety)\}$). Notice that this situation would not arise in the case of birds that are not pigeons, thus producing the intuitive result.

Example 2 (Based on [11]). Species in the phylum of Mollusca, which includes clams and snails, normally have external shells. Mollusca contains the class Cephalopoda, which includes squid and octopi. Cephalopods normally lack external shells; however they include the genus Nautilus, and nautiloids normally have external shells. Formally, we have $\Sigma_2 = (T_S, T_D, A)$ where:

$$T_S = \left\{ \begin{array}{l} \mathsf{Nautilus} \sqsubseteq \mathsf{Cephalopoda} \\ \mathsf{Cephalopoda} \sqsubseteq \mathsf{Mollusca} \end{array} \right\} \; T_D = \left\{ \begin{array}{l} \mathsf{Nautilus} \sqsubseteq \mathsf{External\_shell} \\ \mathsf{Mollusca} \sqsubseteq \mathsf{External\_shell} \\ \mathsf{Cephalopoda} \sqsubseteq \neg \mathsf{External\_shell} \end{array} \right\}$$

$$A = \left\{ \mathsf{a} : \mathsf{Nautilus} \right\}$$

Consider the query "Is the chambered nautilus likely to have an external shell?" First, because nautiloids normally have external shells, we could answer "Yes". Second, since nautiloids are cephalopods and since cephalopods generally lack external shells, we maybe could affirm "No". Third, because cephalopods are mollusks, which generally have external shells, maybe "Yes" is correct after all. This ontology is interpreted as the DeLP program:

$$\Pi = \left\{ \begin{array}{l} cephalopoda(X) \leftarrow nautilus(X). \\ mollusca(X) \leftarrow cephalopoda(X). \\ \sim nautilus(X) \leftarrow\!\sim cephalopoda(X). \\ \sim cephalopoda(X) \leftarrow\!\sim mollusca(X). \\ nautilus(a). \end{array} \right\}$$

$$\Delta = \left\{ \begin{array}{l} external\_shell(X) \multimap\!\prec nautilus(X). \\ external\_shell(X) \multimap\!\prec mollusca(X). \\ \sim external\_shell(X) \multimap\!\prec cephalopoda(X). \end{array} \right\}$$

In this case, there are two arguments supporting opposing conclusions: $\mathcal{A}$ supporting $external\_shell(a)$ and $\mathcal{B}$ supporting $\sim external\_shell(a)$, where $\mathcal{A} = \{external\_shell(a) \multimap\!\prec nautilus(a)\}$ and $\mathcal{B} = \{\sim external\_shell(a) \multimap\!\prec cephalopoda(a)\}$. Here $\mathcal{A}$ is defeated by $\mathcal{B}$ because $\mathcal{B}$ is more specific than $\mathcal{A}$.

Example 3 (Based on [11]). Mammals usually do not fly. Bats normally fly. Vampires are mammals. Bats are mammals. Given that Dracula is a vampire, what should we believe about Dracula's ability to fly? Given that Tea Tray is a bat, should we believe that Tea Tray flies? This situation is represented by $\Sigma_3 = (T_S, T_D, A)$ where:

$$T_S = \left\{ \begin{array}{l} \mathsf{Vampire} \sqsubseteq \mathsf{Mammal} \\ \mathsf{Bat} \sqsubseteq \mathsf{Mammal} \end{array} \right\} \; T_D = \left\{ \begin{array}{l} \mathsf{Mammal} \sqsubseteq \neg \mathsf{Fly} \\ \mathsf{Bat} \sqsubseteq \mathsf{Fly} \end{array} \right\}$$

$$A = \left\{ \begin{array}{l} \mathsf{DRACULA} : \mathsf{Vampire} \\ \mathsf{TEA\_TRAY} : \mathsf{Bat} \end{array} \right\}$$

This ontology is interpreted as $\mathcal{P}_3 = (\Pi, \Delta)$:

$$\Pi = \left\{ \begin{array}{l} mammal(X) \leftarrow vampire(X). \\ mammal(X) \leftarrow bat(X). \\ \sim vampire(X) \leftarrow \sim mammal(X). \\ \sim bat(X) \leftarrow \sim mammal(X). \\ vampire(dracula). \\ bat(tea\_tray). \end{array} \right\} \quad \Delta = \left\{ \begin{array}{l} \sim fly(X) \prec\!\!\!- mammal(X). \\ fly(X) \prec\!\!\!- bat(X). \end{array} \right\}$$

If we ask if Dracula flies, DeLP finds an undefeated argument $\mathcal{A}$ for $\sim fly(dracula)$, where $\mathcal{A} = \{\sim fly(dracula) \prec\!\!\!- mammal(dracula)\}$, stating that Dracula does not fly because he is a mammal as he is a vampire. So the DRACULA is a strict member of ¬Fly. When we query if Tea Tray can fly, we get a similar argument but this one is defeated by another argument that says that Tea Tray can fly because he is a bat and bats fly. Formally, $\langle \mathcal{C}, \sim fly(tea\_tray) \rangle$ is defeated by $\langle \mathcal{D}, fly(tea\_tray) \rangle$, where: $\mathcal{C} = \{\sim fly(tea\_tray) \prec\!\!\!- mammal(tea\_tray)\}$ and $\mathcal{D} = \{fly(tea\_tray) \prec\!\!\!- bat(tea\_tray)\}$.

Example 4 (Based on [11], who originally took it from [16]). Utah residents are usually Mormons. Mormons usually do not drink beer. BYU alumni living in Utah are usually football fans. Football fans usually drink beer. Formally, we must code this knowledge as $\Sigma_4 = (\emptyset, T_D, A)$ where:

$$T_D = \left\{ \begin{array}{l} \mathsf{Lives\_in\_utah} \sqsubseteq \mathsf{Mormon} \\ \mathsf{Mormon} \sqsubseteq \neg \mathsf{Drinks} \\ \mathsf{Football\_fan} \sqsubseteq \mathsf{Drinks} \\ \mathsf{BYU\_Alumni} \sqsubseteq \mathsf{Football\_fan} \end{array} \right\}$$

$$A = \{ \mathsf{a} : \mathsf{BYU\_Alumni} \qquad \mathsf{a} : \mathsf{Lives\_in\_utah}. \}$$

Should we believe that BYU alumni living in Utah drink beer? The interpretation of Σ_4 is $\mathcal{P}_4 = (\Pi, \Delta)$ where:

$$\Pi = \{ byu\_alumni(a). \qquad lives\_in\_utah(a). \}$$

$$\Delta = \left\{ \begin{array}{l} mormon(X) \prec\!\!\!- lives\_in\_utah(X). \\ \sim drinks(X) \prec\!\!\!- mormon(X). \\ drinks(X) \prec\!\!\!- football\_fan(X). \\ football\_fan(X) \prec\!\!\!- byu\_alumni(X). \end{array} \right\}$$

This case presents the same pattern of the Nixon's diamond in the sense that it propagates undecidability. We can find two equally strong arguments: $\langle \mathcal{A}, drinks(a) \rangle$ and $\langle \mathcal{B}, \sim drinks(a) \rangle$ where

$$\mathcal{A} = \left\{ \begin{array}{l} drinks(a) \prec\!\!\!- football\_fan(a) \\ football\_fan(a) \prec\!\!\!- byu\_alumni(a) \end{array} \right\}$$

$$\mathcal{B} = \left\{ \begin{array}{l} \sim drinks(a) \prec\!\!\!- mormon(a) \\ mormon(a) \prec\!\!\!- lives\_in\_utah(a) \end{array} \right\}$$

Therefore the membership of a to Drinks is undecided.

The next example is a classical one that Caminada and Amgoud refer to as a situation that presents problems in certain argumentation systems because it yields to unintuitive results such us warranting literals that should not (see [14]).

Example 5 (Based on [14]). Let $\Sigma_5 = (T_S, T_D, A)$ be a δ-ontology where:

$$T_S = \left\{ \begin{array}{l} \mathsf{Bachelor} \sqsubseteq \neg \mathsf{HasWife}; \\ \mathsf{Married} \sqsubseteq \mathsf{HasWife} \end{array} \right\}$$
$$T_D = \left\{ \begin{array}{l} \mathsf{HasWeddingRing} \sqsubseteq \mathsf{Married}; \\ \mathsf{GoesOutLate} \sqsubseteq \mathsf{Bachelor} \end{array} \right\}$$
$$A = \left\{ \begin{array}{l} \mathsf{JOHN} : \mathsf{HasWeddingRing}; \\ \mathsf{JOHN} : \mathsf{GoesOutLate} \end{array} \right\}$$

It says that someone who is a bachelor does not have a wife and somebody who is married do have a wife; usually someone who wears a wedding ring is married and someone who is a bachelor goes out at night, and it is known that John wears a wedding ring and goes out at night. The DeLP program $\mathcal{P}_5 = (\Pi, \Delta)$ that represents Σ_5 is:

$$\Pi = \left\{ \begin{array}{l} hasWeddingRing(john); \\ goesOutLate(john); \\ \sim hasWife(X) \leftarrow bachelor(X); \\ hasWife(X) \leftarrow married(X) \end{array} \right\}$$
$$\Delta = \left\{ \begin{array}{l} married(X) \multimap hasWeddingRing(X); \\ bachelor(X) \multimap goesOutLate(X) \end{array} \right\}$$

The answer to the query $bachelor(john)$ is *Undecided* because the argument structure $\langle \mathcal{A}, bachelor(john) \rangle$ is defeated by $\langle \mathcal{B}, hasWife(john) \rangle$ where [1]

$$\mathcal{A} = \{ bachelor(john) \multimap goesOutLate(john) \}$$
$$\mathcal{B} = \left\{ \begin{array}{l} hasWife(john) \leftarrow married(john); \\ married(john) \multimap hasWeddingRing(john) \end{array} \right\}.$$

Likewise, the answer to the query $married(john)$ is *Undecided* because the argument $\langle \mathcal{C}, married(john) \rangle$ is defeated by $\langle \mathcal{D}, \sim hasWife(john) \rangle$ where:

$$\mathcal{C} = \{ married(john) \multimap hasWeddingRing(john) \}$$
$$\mathcal{D} = \left\{ \begin{array}{l} \sim hasWife(john) \leftarrow bachelor(john); \\ bachelor(X) \multimap goesOutLate(john) \end{array} \right\}.$$

The answers to the queries $hasWife(john)$ and $\sim hasWife(john)$ are Undecided as well because $\langle \mathcal{E}, hasWife(john) \rangle$ and $\langle \mathcal{F}, \sim hasWife(john) \rangle$ are blocking defeaters of each other, where:

$$\mathcal{E} = \left\{ \begin{array}{l} hasWife(john) \leftarrow married(john) \\ married(john) \multimap hasWeddingRing(john) \end{array} \right\}$$
$$\mathcal{F} = \left\{ \begin{array}{l} \sim hasWife(john) \leftarrow bachelor(john); \\ bachelor(john) \multimap goesOutLate(john) \end{array} \right\}.$$

As noted by Caminada and Amgoud [14], DeLP handles this situation correctly, so the instance checking problems $\mathsf{JOHN} : \mathsf{Married}$ and $\mathsf{JOHN} : \mathsf{Bachelor}$ are undecided and no counterintuitive solution is inferred in the δ-ontologies framework.

4 Discussion and Conclusions

In Figure 1, we present a revised version of the algorithm presented in [7] for computing instance checking of possibly inconsistent DL ontologies. In this version,

[1] Notice that we abuse the notation here and add strict rules in arguments for emphasis.

when the ontology is both consistent and coherent, we just use the RacerPro reasoner to compute instance checking; otherwise, we rely on DeLP to compute answers. Besides, to translate from DL to Prolog, we rely on the `dlpconvert` tool (see [17]) and from reasoning in DeLP in the *DeLP Server* application (see [18]). In [7] we decided to translate all ontologies into DeLP. In our opinion, the version presented in this paper is an improvement because not all ontologies can be translated to DeLP. Notice that this algorithm shows influences from [5] as they rely on paraconsistent logic reasoning only when ontologies are inconsistent.

Algorithm AnswerQueryFromOntology(DL Ontology Σ, Query $C(a)$)
Submit Σ and $C(a)$ to RacerPro DL reasoner
If Σ is consistent and coherent **Then**
  print RacerPro answer
Else
  Submit Σ to `dlpconvert` to get Prolog program $\mathcal{P}_{prolog}$
  If Σ could be translated into Prolog **Then**
    Convert $\mathcal{P}_{prolog}$ to get DeLP program $\mathcal{P}$
    Submit $Q(a)$ and $\mathcal{P}$ to the DeLP Server
    If DeLP's answer is *Yes* **Then**
      If the argument for $C(a)$ is empty **Then**
        Print a is a strict member of C
      Else
        Print a is a justified member of C
      End If
    Else
      If DeLP's answer for $C(a)$ is *No* **Then**
        If the argument for $\sim C(a)$ is empty **Then**
          Print a is a strict member of $\neg C$
        Else
          Print a is a justified member of $\neg C$
        End If
      Else
        If DeLP's answer for $C(a)$ is *Undecided* **Then**
          Print the membership of a to C is undecided
        End If
      End If
    End If
  End If
End If
End Algorithm

Fig. 1. Revised algorithm for instance checking

We presented some classical examples in the area of inheritance networks and what the behavior of the δ-ontologies framework is when presented with them. As our current reasearch work, we are considering the implementation of the

framework proposed and contrasting its performance against the complexity of real-world ontologies.

Acknowledgments: This research is funded by Secretaría General de Ciencia y Técnica, Universidad Nacional del Sur, Argentina.

References

1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P., eds.: The Description Logic Handbook — Theory, Implementation and Applications. Cambridge University Press (2003)
2. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American (2001)
3. Haarslev, V., Möller, R.: RACER System Description. Technical report, University of Hamburg, Computer Science Department (2001)
4. Parsia, B., Sirin, E.: Pellet: An OWL DL Reasoner. In: 3rd International Semantic Web Conference (ISWC2004). (2004)
5. Huang, Z., van Harmelen, F., ten Teije, A.: Reasoning with Inconsistent Ontologies. In Kaelbling, L.P., Saffiotti, A., eds.: Proc. 19th International Joint Conference on Artificial Intelligence (IJCAI'05), Edinburgh, Scotland (August 2005) 454–459
6. Ribeiro, M.M., Wassermann, R.: Base revision for ontology debugging. J. Log. Comput. **19**(5) (2009) 721–743
7. Gómez, S.A., Chesñevar, C.I., Simari, G.R.: Reasoning with Inconsistent Ontologies Through Argumentation. Applied Artificial Intelligence **1**(24) (2010) 102–148
8. Gómez, S.A., Chesñevar, C.I., Simari, G.R.: ONTOarg: A Decision Support Framework for Ontology Integration based on Argumentation. Expert Systems with Applications **40** (2013) 1858–1870
9. Gómez, S.A., Simari, G.R.: Merging of ontologies using belief revision and defeasible logic programming. Inteligencia Artificial **16**(52) (2013) 16–28
10. Bench-Capon, T.J.M., Dunne, P.E.: Argumentation in artificial intelligence. Artificial Intelligence **171**(10-15) (2007) 619–641
11. Bender, E.: Mathematical Methods in Artificial Intelligence. IEEE Computer Society Press (1996)
12. García, A., Simari, G.: Defeasible Logic Programming an Argumentative Approach. Theory and Practice of Logic Programming **4**(1) (2004) 95–138
13. 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)
14. Caminada, M., Amgoud, L.: On the evaluation of argumentation formalisms. Artificial Intelligence **171** (2007) 286–310
15. Horty, J.F., Thomasson, R.H., Touretzky, D.S.: A skeptical theory of inheritance in nonmonotonic semantic networks. Artificial Intelligence (42) (1990) 311–348
16. Nute, D.: Basic defeasible logic. Intensional Logics for Programming (1992) 125–154
17. Motik, B., Vrandecić, D., Hitzler, P., Sure, Y., Studer, R.: dlpconvert: Converting OWL DLP statements to logic programs. In: European Semantic Web Conference 2005 (ESWC 2005) Demos and Posters. (2005)
18. García, A.J., Rotstein, N.D., Tucat, M., Simari, G.R.: An Argumentative Reasoning Service for Deliberative Agents. In Zhang, Z., Siekmann, J.H., eds.: KSEM. Volume 4798 of Lecture Notes in Computer Science., Springer (2007) 128–139