Towards a Practical Implementation of a Reasoner for Inconsistent Possibilistic Description Logic Programming Ontologies

Sergio Alejandro Gómez

Artificial Intelligence Research and Development Laboratory (LIDIA)
Instituto de Ciencias e Ingeniería de la Computación (ICIC, Conicet, UNS)
Department of Computer Science and Engineering
Universidad Nacional del Sur
Av. Alem 1253, Bahía Blanca, Argentina
Email: sag@cs.uns.edu.ar

Abstract. This work reports on our efforts to implement a practical reasoner based on Dung-style argumentation semantics for potentially inconsistent possibilistic ontologies. Our Java-based implementation targets a subset of the description logic programming fragment that we codify in a Racer-like syntax suitably adapted for representing certainty degrees of both axioms and assertions. We introduce our approach with a running example, discuss implementation issues and present time complexity results.

Keywords: Ontology reasoning with inconsistency, possibilistic description logic programming ontologies, argumentation semantics

1 Introduction and Motivations

Ontologies as agreed representations of a problem domain play an important role in the implementation of the Semantic Web. They are defined by a knowledge engineer from scratch or importing some parts of it from reputable sources, usually in languages based on Description Logics [2]. When an ontology is inconsistent (a situation characterized by a logical contradiction), ontology reasoners (e.g. Racer or Pellet) can normally pinpoint the source of inconsistency but rely on the knowledge engineer to repair the ontology (i.e. making it consistent again). But many times the knowledge engineer does not have the authority to correct the inconsistencies because he cannot have the expertise or the field being modeled can be intrinsically inconsistent.

Instead of repairing an inconsistent ontology, either manually or automatically (e.g. with belief revision), we believe in using alternate approaches to reasoning that can cope with inconsistencies and yet entail meaningful conclusions. In this regard, defeasible argumentation [3] is an approach to non-monotonic reasoning that presents an elegant form of entailing conclusions from inconsistent knowledge bases. Instead of relying in the notion of proof, argumentation relies in the notion of argument (i.e. a set of reasons supporting a defeasible

conclusion). One argument attacks another argument when the former disagrees with some assumption, intermediate or final conclusion of the latter, thus taking into account the internal structure of the arguments or the relative weights of their supporting reasons.

In this paper, we report on the Java-based implementation of a system using structured argumentation that allows us to reason on possibly inconsistent possibilistic ontologies specified in a Racer-like syntax. In our approach, the ontologies are interpreted as possibilistic logic programs under a Dung's semantics. The elements that we consider are: a language for defining ontologies whose axioms and assertions are weighted with a certainty degree, structured arguments based on the language of logic programming with weights (viz. possibilistic logic programming), a deductive relation that allows to build arguments, a form of computing attacks between pairs of arguments supporting contradicting conclusions, a way to compute a Dung's argumentation framework graph, a way to compute the extension that can be inferred from such graph that ultimately characterizes the conclusions that can be obtained from the input ontology. Our results are applicable to Semantic Web reasoners based on Description Logic languages.

The rest of this paper is structured as follows. In Sect. 2 we recall Possibilistic Description Logic Ontologies. In Sect. 3 we briefly explain reasoning in argumentation frameworks. In Sect. 4 we explain how to relate inference tasks in Possibilistic Description Logic Ontologies with reasoning in argumentation frameworks with structured arguments. In Sect. 5 we introduce a prototype implementation for the reasoning framework presented. In Sect. 6 we review related work. Finally in Sect. 7, we conclude and discuss future work.

2 Possibilistic Description Logic Programming Ontologies

We review here possibilistic description logic ontologies [7], that are ontologies with numeric degrees of certainty attached to both axioms and assertions. We briefly recall reasoning in description logics along with the variations needed for reasoning with possibilistic description logics.

2.1 A Brief Recall of Description Logics

Description Logics (DL) are a well-known family of knowledge representation formalisms [2]. 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 ($\neg C$), existential 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. The symbol $\bot$ stands for the empty concept.

A DL ontology consists of two finite and mutually disjoint sets: a *Tbox* which introduces the *terminology* and an *Abox* (assertional box) which contains facts about particular objects in the application domain. A Tbox contains inclusion axioms $C \sqsubseteq D$, where C and D are (possibly complex) concept descriptions, meaning that every individual of C is also a D, and equality axioms $C \equiv D$ meaning that C and D are equivalent concepts (i.e. every individual in C is an individual in D and vice versa). Objects in the Abox are referred to by a finite number of *individual names* and these names may be used in assertional statements: *concept assertions* of two types: $a : C$ (meaning the individual a is a member of concept C), and *role assertions* of the type $\langle a, b\rangle : R$ (meaning that a is related to b through the role R).

Many reasoning Tbox and Abox reasoning tasks are defined for DL ontologies (see [2]), but in this work we are only interested in *instance checking* that refers to determining if an individual is a member of a certain concept.

One form of assigning semantics to a DL ontology is based on the fact that DL is isomorphic with first-order logic restricted to two variables. Then, for example, the inclusion axiom $C \sqsubseteq D$ can be interpreted as the first-order logic formula $(\forall x)(c(x) \rightarrow d(x))$ and the assertion $a : C$ as $c(a)$. Description Logic Programming (DLP) approaches [11] take advantage of this to interpret those expressions as the Prolog rules "`d(X) :- c(X).`" and "`c(a).`", resp.

2.2 Fundamentals of Possibilistic Description Logics

We now recall the fundamentals of possibilistic description logic ontologies. Our presentation is based on [4, 7]. Let $\mathcal{L}_{DL}$ be a DL description language, a *possibilistic DL ontology* is a set of possibilistic axioms of the form $(\varphi, W(\varphi))$ where φ is an axiom expressed in $\mathcal{L}_{DL}$ and $W(\varphi) \in [0, 1]$ is the degree of certainty (or priority) of φ. Namely, a possibilistic DL ontology Σ is such that $\Sigma = \{(\varphi_i, W(\varphi_i)) : i = 1, \ldots, n\}$. Only somewhat certain information is explicitly represented in a possibilistic ontology. That is, axioms with a null weight ($W(\varphi) = 0$) are not explicitly represented in the knowledge base. The weighted axiom $(\varphi, W(\varphi))$ means that the certainty degree of φ is at least equal to $W(\varphi)$. A possibilistic DL ontology Σ will also be represented by a pair $\Sigma = (T, A)$ where elements in both T and A may be uncertain. Note that if we consider all $W(\varphi_i) = 1$, then we find a classical DL ontology $\Sigma^* = \{\varphi_i : (\varphi_i, W(\varphi_i)) \in \Sigma\}$. We say that Σ is consistent if the classical ontology obtained from Σ by ignoring the weights associated with axioms is consistent, and inconsistent otherwise. Notice that the weights $W(\cdot)$ for axioms must be provided by the knowledge engineer that designs the knowledge base, thus specifying the relative importance of rules and facts.

Example 1. Let $\Sigma_1 = (T, A)$ be the ontology modeling a variation of the famous Tweety example from the non-monotonic literature. It expresses that a bird generally flies, all penguins are birds, penguins do not usually fly, birds with broken wings normally do not fly either, and pilots can almost always fly. It is known that Tweety is a penguin with almost certainly a broken wing and most

likely a pilot. Formally:

$$T = \left\{ \begin{array}{lll} (\mathsf{Bird} \sqsubseteq \mathsf{Flies}, 0.6), & (\mathsf{Penguin} \sqsubseteq \mathsf{Bird}, 1.0), & (\mathsf{Pilot} \sqsubseteq \mathsf{Flies}, 0.9) \\ (\mathsf{Penguin} \sqsubseteq \neg\mathsf{Flies}, 0.8), & (\mathsf{Bird} \sqcap \mathsf{BrokenWing} \sqsubseteq \neg\mathsf{Flies}, 0.7), & \end{array} \right\}$$

$$A = \left\{ \begin{array}{ll} (\mathsf{TWEETY} : \mathsf{BrokenWing}, 0.8), & (\mathsf{TWEETY} : \mathsf{Penguin}, 1.0), \\ (\mathsf{TWEETY} : \mathsf{Pilot}, 0.9) & \end{array} \right\}$$

In Σ_1^*, because Tweety is both a bird and a penguin, he is both a member of Flies and $\neg\mathsf{Flies}$, meaning that he is a member of $\bot$. Traditional reasoners are not able to infer anything from such an inconsistent ontology, thus invalidating even reasoning with consistent parts of the offending ontology.

3 A Brief Introduction to Argumentation

Here we recall the basic notions of non-monotonic reasoning with argumentation.

3.1 Dung-Style Abstract Argumentation

Abstract argumentation frameworks do not presuppose any internal structure of arguments, thus considering only the interactions of arguments by means of an attack relation between arguments (we base on [14], that in turn bases on [6]).

Definition 1. *An* abstract argumentation framework $\mathcal{AF}$ *is a pair* $(Arg, \rightarrow)$ *where* Arg *is a set of arguments and* $\rightarrow$ *is a relation of* Arg *into* Arg*.*

In this work, we will consider only finitary argumentation systems (i.e. argument systems with a finite number of arguments). For two arguments $\mathcal{A}$ and $\mathcal{B}$ in Arg, the relation $\mathcal{A} \rightarrow \mathcal{B}$ means that the argument $\mathcal{A}$ attacks the argument $\mathcal{B}$. Abstract argumentation frameworks can be concisely represented by directed graphs, where arguments are represented as nodes and edges model the attack relation.

Example 2. Consider the argumentation framework $\mathcal{AF}_2 = (Arg, \rightarrow)$ where $Arg = \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4, \mathcal{A}_5, \mathcal{A}_6, \mathcal{A}_7, \mathcal{A}_8\}$ and $\rightarrow = \{(\mathcal{A}_5, \mathcal{A}_8), (\mathcal{A}_6, \mathcal{A}_8), (\mathcal{A}_7, \mathcal{A}_5), (\mathcal{A}_7, \mathcal{A}_6)\}$. The framework is shown graphically in Fig. 1 and, although it is not necessary from a mathematical viewpoint, we can assign meaning to the above arguments to provide some intuition:

- $\mathcal{A}_1$: Tweety has a broken wing
- $\mathcal{A}_2$: Tweety is a penguin
- $\mathcal{A}_3$: Tweety is a pilot[1]
- $\mathcal{A}_4$: Tweety is a bird
- $\mathcal{A}_5$: Tweety does not fly because he is a penguin and penguins do not usually fly
- $\mathcal{A}_6$: Tweety does not fly because he has a broken wing
- $\mathcal{A}_7$: Tweety flies because he is also a pilot
- $\mathcal{A}_8$: Tweety flies because he is a bird and birds normally fly

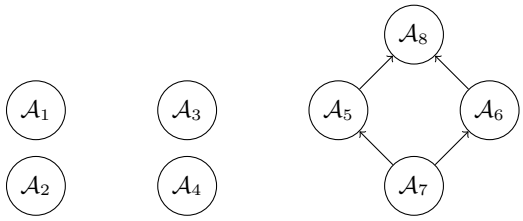

Fig. 1. Abstract argumentation framework presented in Ex. 2

Semantics are usually given to abstract argumentation frameworks by means of extensions. An extension E of an argumentation framework $\mathcal{AF} = (Arg, \rightarrow)$ is subset of Arg that gives some coherent view on the argumentation underlying $\mathcal{AF}$. In this work, we will reason under grounded semantics despite that other semantics have been proposed.

Definition 2. *Let $\mathcal{AF} = (Arg, \rightarrow)$ be an argumentation framework.*

- *An extension $E \subseteq Arg$ is* conflict-free *iff there are no $\mathcal{A}, \mathcal{B} \in E$ with $\mathcal{A} \rightarrow \mathcal{B}$.*
- *An argument $\mathcal{A} \in Arg$ is* acceptable *with respect to an extension $E \subseteq Arg$ iff for every $\mathcal{B} \in Arg$ with $\mathcal{B} \rightarrow \mathcal{A}$ there is a $\mathcal{A}' \in E$ with $\mathcal{A}' \rightarrow \mathcal{B}$.*
- *An extension $E \subseteq Arg$ is* admissible *iff it is conflict free and all $\mathcal{A} \in E$ are acceptable with respect to E.*
- *An extension $E \subseteq Arg$ is* complete *iff it is admissible and there is no $\mathcal{A} \in Arg \backslash E$ that is acceptable with respect to E.*
- *An extension $E \subseteq Arg$ is* grounded *iff it is complete and E is minimal with respect to set inclusion.*

The intuition behind admissibility is that an argument can only be accepted if there are no attackers that are accepted and if an argument is not accepted then there has to be an acceptable argument attacking it. The idea underlying the completeness property is that all acceptable arguments should be accepted.

The grounded extension is the minimal set of acceptable arguments and is uniquely determined. It can be computed as follows: first, all arguments that have no attackers are added to the empty extension E and those arguments and all arguments that are attacked by one of these arguments are removed from the framework; then the process is repeated; if one obtains a framework where there are no unattacked arguments, the remaining arguments are also removed.

Example 3. Consider again the argumentation framework $\mathcal{AF}_2$ presented in Ex. 2. The grounded extension E of $\mathcal{AF}_2$ is given by $E = \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4, \mathcal{A}_7, \mathcal{A}_8\}$.

We will present a reasoning framework whose underlying interpretation will coincide exactly with the behavior of the framework above, thus allowing to reason on the contents of the ontology in Ex. 1. We will also describe a reasoning system that implements this approach.

[1] We think of the Madagascar movies to motivate this example (See `https://en.wikipedia.org/wiki/Madagascar:_Escape_2_Africa`).

3.2 Possibilistic Logic Programming

The P-DeLP [1] language $\mathcal{L}$ is defined from a set of ground fuzzy atoms (fuzzy propositional variables) $\{p, q, \ldots\}$ together with the connectives $\{\sim, \wedge, \leftarrow\}$. The symbol $\sim$ stands for *negation*. A literal $L \in \mathcal{L}$ is a ground (fuzzy) atom $\sim q$, where q is a ground (fuzzy) propositional variable. A *rule* in $\mathcal{L}$ is a formula of the form $Q \leftarrow L_1 \wedge \ldots \wedge L_n$, where $Q, L_1, \ldots, L_n$ are literals in $\mathcal{L}$. When $n = 0$, the formula $Q \leftarrow$ is called a *fact*. The term *goal* will refer to any literal $Q \in \mathcal{L}$. Facts, rules and goals are the well-formed formulas in $\mathcal{L}$.

Definition 3. *A* certainty-weighted clause*, or simply weighted clause, is a pair* (φ, α)*, where* φ *is a formula in* $\mathcal{L}$ *and* $\alpha \in [0, 1]$ *expresses a lower bound for the certainty of* φ *in terms of a necessity measure.*

The original P-DeLP language is based on Possibilistic Gödel Logic or PGL, which is able to model both uncertainty and fuzziness and allows for a partial matching mechanism between fuzzy propositional variables. For simplicity, Chesñevar et al. [1] restrict themselves to the fragment of P-DeLP built on non-fuzzy propositions, and hence based on the necessity-valued classical propositional Possibilistic logic. As a consequence, possibilistic models are defined by possibility distributions on the set of classical interpretations and the proof method for P-DeLP formulas, written $\vdash$, is defined based on the generalized modus ponens rule, that from $(L_0 \leftarrow L_1 \wedge \ldots \wedge L_k, \gamma)$ and $(L_1, \beta_1), \ldots, (L_k, \beta_k)$ allows to infer $(L_0, \min(\gamma, \beta_1, \ldots, \beta_k))$, which is a particular instance of the possibilistic resolution rule, and which provides the *non-fuzzy* fragment of P-DeLP with a complete calculus for determining the maximum degree of possibilistic entailment for weighted literals.

In P-DeLP *certain* and *uncertain* clauses can be distinguished. A clause (φ, α) is referred as certain if $\alpha = 1$ and uncertain otherwise. A set of clauses Γ is deemed as *contradictory*, denoted $\Gamma \vdash \bot$, when $\Gamma \vdash (q, \alpha)$ and $\Gamma \vdash (\sim q, \beta)$, with $\alpha > 0$ and $\beta > 0$, for some atom in $\mathcal{L}$. A P-DeLP program is a set of weighted rules and facts in $\mathcal{L}$ in which certain and uncertain information is distinguished. As an additional requirement, certain knowledge is required to be non-contradictory. Formally:

Definition 4. *A P-DeLP program* $\mathcal{P}$ *(or just* program $\mathcal{P}$*) is a pair* (Π, Δ)*, where* Π *is a non-contradictory finite set of certain clauses, and* Δ *is a finite set of uncertain clauses.*

Definition 5. *Given a program* $\mathcal{P} = (\Pi, \Delta)$*, a set* $\mathcal{A} \subseteq \Delta$ *of uncertain clauses is an* argument *for a goal* Q *with necessity degree* $\alpha > 0$*, denoted* $\langle A, Q, \alpha \rangle$*, iff: (i)* $\Pi \cup \mathcal{A} \vdash (Q, \alpha)$*; (ii)* $\Pi \cup \mathcal{A}$ *is non-contradictory, and (iii) there is no* $\mathcal{A}_1 \subset \mathcal{A}$ *such that* $\Pi \cup \mathcal{A}_1 \vdash (Q, \beta)$*,* $\beta > 0$*. Let* $\langle \mathcal{A}, Q, \alpha \rangle$ *and* $\langle \mathcal{S}, R, \beta \rangle$ *be two arguments,* $\langle \mathcal{S}, R, \beta \rangle$ *is a* subargument *of* $\langle \mathcal{A}, Q, \alpha \rangle$ *iff* $\mathcal{S} \subseteq \mathcal{A}$*.*

Conflict among arguments is formalized by the notions of counterargument and defeat.

Definition 6. *Let $\mathcal{P}$ be a program, and let $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ and $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ be two arguments in $\mathcal{P}$. We say that $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$* counterargues *$\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ iff there exists a subargument (called disagreement subargument) $\langle \mathcal{S}, Q, \beta \rangle$ of $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ such that $\Pi \cup \{(Q_1, \alpha_1), (Q, \beta)\}$ is contradictory. The literal (Q, β) is called* disagreement literal*.*

Defeat among arguments involves the consideration of *preference criteria* defined on the set of arguments. The criterion applied here will be defined on the basis of necessity measures associated with arguments.

Definition 7. *Let $\mathcal{P}$ be a program, and let $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ and $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ be two arguments in $\mathcal{P}$. We will say that $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ is a* defeater *for $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ iff $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ counterargues argument $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ with disagreement subargument $\langle \mathcal{A}, Q, \alpha \rangle$, with $\alpha_1 \geq \alpha$. If $\alpha_1 > \alpha$ then $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ is called a* proper defeater*, otherwise ($\alpha_1 = \alpha$) it is called a* blocking defeater*.*

Notice that we digress from the original P-DeLP formalism in that (i) we include facts in the support of arguments and (ii) facts are allowed to have a weight different than one (so allowing them to be considered as presumptions).

Example 4. Consider again the ontology Σ_1 presented in Ex. 1. This ontology is interpreted as the equivalent possibilistic program $\mathcal{P}_1$ where:

$$\mathcal{P}_1 = \left\{ \begin{array}{ll} (brokenWing(tweety), 0.8), & (penguin(tweety), 1.0), \\ (pilot(tweety), 0.9), & \\ (flies(X) \leftarrow bird(X), 0.6), & \\ (bird(X) \leftarrow penguin(X), 1.0), & \\ (\sim flies(X) \leftarrow penguin(X), 0.8), & \\ (\sim flies(X) \leftarrow bird(X), brokenWing(X), 0.7), & \\ (flies(X) \leftarrow pilot(X), 0.9) & \end{array} \right\}.$$

Exactly 8 arguments can be built from this knowledge base (notice that they coincide with the ones presented in Ex. 2):

- $\langle \mathcal{A}_1, brokenWing(tweety), 0.8 \rangle$ where $\mathcal{A}_1 = \{ (brokenWing(tweety), 0.8) \}$
- $\langle \mathcal{A}_2, penguin(tweety), 1.0 \rangle$ where $\mathcal{A}_2 = \{ (penguin(tweety), 1.0) \}$
- $\langle \mathcal{A}_3, pilot(tweety), 0.9 \rangle$ where $\mathcal{A}_3 = \{ (pilot(tweety), 0.9) \}$
- $\langle \mathcal{A}_4, bird(tweety), 1.0 \rangle$ where $\mathcal{A}_4 = \{ (bird(tweety) \leftarrow penguin(tweety), 1.0) \} \cup \mathcal{A}_2$
- $\langle \mathcal{A}_5, \sim flies(tweety), 0.8 \rangle$ where

$$\mathcal{A}_5 = \{ (\sim flies(tweety) \leftarrow penguin(tweety), 0.8) \} \cup \mathcal{A}_2$$

- $\langle \mathcal{A}_6, \sim flies(tweety), 0.7 \rangle$ where

$$\mathcal{A}_6 = \{ (\sim flies(tweety) \leftarrow bird(tweety), brokenWing(tweety), 0.7) \} \cup \mathcal{A}_1 \cup \mathcal{A}_4$$

- $\langle \mathcal{A}_7, flies(tweety), 0.9 \rangle$ where $\mathcal{A}_7 = \{ (flies(tweety) \leftarrow pilot(tweety), 0.9) \} \cup \mathcal{A}_3$
- $\langle \mathcal{A}_8, flies(tweety), 0.6 \rangle$ where $\mathcal{A}_8 = \{ (flies(tweety) \leftarrow bird(tweety), 0.6) \} \cup \mathcal{A}_4$

The attacks among these arguments are exactly those presented in Fig. 1. Notice that here the attacks are made into final conclusions (thus they are direct attacks). Nonetheless the reasoning framework presented here and the application we built also allow for modeling attacks into premises (i.e. indirect attacks).

4 Reasoning with Inconsistent Ontologies with Argumentation

Here we discuss how to perform reasoning with inconsistent ontologies relating the language of possibilistic description logic with P-DeLP.

4.1 Expressing Possibilistic Ontologies as Possibilistic Logic Programs

Grosof et al. [11] provide a way of expressing a subset of Description Logic ontologies in logic programming, namely the description logic programming subset of DL that can be expressed as Horn knowledge bases. The idea consists of expressing both DL assertional statements and terminological axioms as equivalent Horn clauses. We will explain only the part of the algorithm relevant to this work.

Given an ontology $\Sigma = (T, A)$, for every terminological axiom or assertional statement $(\varphi, W(\phi))$ we will generate a possibilistic axiom $(\mathcal{T}(\varphi), W(\varphi))$, where $\mathcal{T}(\cdot)$ is the transformation function from the language of description logics to the language of Horn clauses. The specification of the $\mathcal{T}$ is as follows: Assertional statements in A of the form $a : C$ are expressed as facts $c(a)$. With respect to T, notice that equivalence axioms $C \equiv D$ can be expressed as two inclusion axioms $C \sqsubseteq D$ and $D \sqsubseteq C$. Then, to successfully express inclusion axioms as Horn-clauses, they first have to be in negation normal form (NNF) (i.e. negation has to be pushed inwards class expressions using well known transformation rules, e.g. De Morgan and double negation laws). Once inclusion axioms are in NNF, axioms $C \sqsubseteq D \sqcap E$ (i.e. with conjunctions in the right-hand side) have to be transformed into $C \sqsubseteq D$ and $C \sqsubseteq E$, and axioms $C \sqcup D \sqsubseteq E$ (i.e. with disjunctions in the left-hand side) are written as $C \sqsubseteq E$ and $D \sqsubseteq E$. Once these steps are performed, we obtain an equivalent ontology composed only of inclusion axioms of the form $C_1 \sqcap \ldots \sqcap C_n \sqsubseteq D$ which are expressed as Horn clauses of the form $d(X) \leftarrow c_1(X), \ldots, c_n(X)$.

Logic programming rules are not equivalent to Horn clauses as the former do not allow to perform contrapositive reasoning. One way to overcome this difficulty is given by computing transposes of rules. As it is usual in non-monotonic reasoning settings, only transposes of strict information (i.e. rules with weight equal to one) are computed because reasoning with transposes of defeasible rules tend to generate fallacious conclusions. The set of transposes of the rule $d(X) \leftarrow c_1(X), \ldots, c_n(X)$ is the set of rules $\{(\sim c_1(X) \leftarrow\sim d(X), \ldots, c_n(X)), \ldots, (\sim c_n(X) \leftarrow c_1(X), \ldots, \sim d(X))\}$.

Example 5 (Continues Ex. 4). Consider again the program $\mathcal{P}_1$ obtained from the ontology Σ_1. In this case, computing the set of transposes of the rule $(bird(X) \leftarrow penguin(X), 1.0)$ allows to add a new rule $(\sim penguin(X) \leftarrow\sim bird(X), 1.0)$ to the program. This new added rule would allow to infer that individuals that are known to be non-birds would also be known to be non-penguins.

4.2 Inference Tasks in Ontologies as Argumentation

Given an possibilistic DL ontology Σ, Σ will be expressed as an equivalent P-DeLP program $\mathcal{P}$. With this program, a grounded extension E will be computed. If $(c(a), \alpha)$ belongs to E then we will say that the individual A is a member of the concept C with a certainty degree α.

Example 6. Recall from Ex. 3 that $E = \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4, \mathcal{A}_7, \mathcal{A}_8\}$. Therefore we can affirm that, from $\mathcal{A}_1$, TWEETY is a member of BrokenWing with certainty level 0.8; from $\mathcal{A}_2$, TWEETY is a member of Penguin with certainty level 1.0; from $\mathcal{A}_3$, TWEETY is a member of Pilot with certainty level 0.9; from $\mathcal{A}_4$, TWEETY is a member of Bird with certainty level 1.0; from $\mathcal{A}_7$, TWEETY is a member of Flies with certainty level 0.9, and from $\mathcal{A}_8$, TWEETY is a member of Flies with certainty level 0.6.

From $\mathcal{A}_7$ and $\mathcal{A}_8$, as Tweety is both a member of Flies with both degrees 0.6 and 0.9, we take a credulous approach considering that TWEETY is a member with degree 0.9.

5 A Prototype Implementation for an Ontology Reasoner Using Dung-Style Argumentation

Here we explain the Java-based implementation we developed to enact the reasoning framework presented above.

5.1 Scripting Languages for Possibilistic Ontologies and Programs

In order to give a text representation for possibilistic DL ontologies, we propose a LISP-like syntax based on the representation language for ontologies of the RACER reasoner. Our proposal permits a knowledge engineer to add labels for specifying the weight of each axiom. Notice as the system presented in this work is a running prototype, we have only implemented the constructors for declaring the signature of the ontology (viz. the `(signature :atomic-concepts` lst_1 `:individuals` lst_2`)` element expressing the list lst_1 of atomic concepts and the list lst_2 of names of individuals), the assertional class statements (viz. the `(instance` a C α`)` element asserting that an individual a is a member of a class C with a certain degree α), and the inclusion axioms (viz. the `(implies` C D α`)` element expressing that a concept C is a subconcept of another concept D with a certain degree α). Besides, only the operators for the complement of a concept (viz. `not`) and conjunction of concepts (viz. `and`) are supported. In Fig. 2.(a), we present the Racer-like script for the ontology of Ex. 1.

Our approach to provide a text representation for possibilistic programs is very straightforward and follows the path marked by DeLP and ASPIC. Facts of the form $(p(a), \alpha)$ are represented as "`p(a) <- true` α" and rules of the form $(p(X) \leftarrow q_1(X), \ldots, q_n(X), \alpha)$ are codified as "`p(X) <- q1(X), ..., qn(X)` α". Besides, the strong negation of `p(X)` is represented with `~p(X)`. In Fig. 2.(b), we present the PDeLP-like script for the program of Ex. 4.

```
(signature
:atomic-concepts (bird penguin flies
       broken_wing pilot)
:individuals (tweety)
)

(instance tweety bird 1.0)
(instance tweety broken_wing 0.8)
(instance tweety penguin 1.0)
(instance tweety pilot 0.9)

(implies bird flies 0.6)
(implies penguin bird 1.0)
(implies penguin (not flies) 0.8)
(implies (and bird broken_wing) (not flies) 0.7)
(implies pilot flies 0.9)
```

(a)

```
broken_wing(tweety) <- true 0.8
penguin(tweety) <- true 1.0
pilot(tweety) <- true 0.9
flies(X) <- bird(X) 0.6
bird(X) <- penguin(X) 1.0
~flies(X) <- penguin(X) 0.8
~flies(X) <- bird(X), broken_wing(X) 0.7
flies(X) <- pilot(X) 0.9
```

(b)

Fig. 2. Scripts for the Tweety ontology and program

5.2 Computing Grounded Extensions for Possibilistic Ontologies

We now give a holistic view of the process to reason with possibly inconsistent possibilistic ontologies. We have implemented a Java-based reasoning engine that allows via a form-based graphical user interface to enter a Racer-like ontology, to see its translation as a P-DeLP program and to see the set of arguments belonging to its uniquely determined grounded extension (see Fig. 3). An executable JAR-file can be downloaded to test the prototype at `http://cs.uns.edu.ar/~sag/engine-v1/`. We discuss the algorithmic details of our implementation here.

Translation of a Racer-like ontology into a possibilistic program: Given a possibilistic ontology Σ, an equivalent possibilistic program $\mathcal{P}$ is obtained using the technique explained in Sect. 4.1.

Propositionalization of the possibilistic program: Given a program $\mathcal{P}$, another program $\mathcal{P}_t$ that contains every rule in $\mathcal{P}$ plus all the transposes of every strict rule in $\mathcal{P}$. As every rule in $\mathcal{P}_t$ is intended as rule-scheme for ground rules (i.e. rules without variables), the next step comprises computing the propositionalization of $\mathcal{P}_t$ called $\mathcal{P}_p$.

Computation of arguments: Once a propositional program $\mathcal{P}_p$ is obtained, the reasoning engine computes the set of arguments that can be derived from $\mathcal{P}_p$. The set of arguments is built inductively. For every fact (or presumption if its weight is less than 1.0), there is a trivial argument whose support is the fact itself and its weight is given by the weight of the fact. For every rule $r = (H \leftarrow B_1, \ldots, B_n, \alpha)$, if there are n arguments $\mathcal{A}_1, \ldots, \mathcal{A}_n$ for $B_1, \ldots, B_n$ with weights $\alpha_1, \ldots, \alpha_n$, respectively, then a new argument for H is added with the n subarguments and the rule r as support and whose weight is the minimum among $\alpha, \alpha_1, \ldots, \alpha_n$. This process continues until a fix-point is reached. Notice that for simplicity, our current implementation does not check for internal consistency in argument construction.

Computing attacks between arguments: The next step involves creating the argumentation framework. In practice, this step requires to create a directed graph $G = (V, E)$ whose vertices V are arguments and the directed edges E represent an attack between a pair of arguments (see Sect. 3.1). Discovering the edges requires iterating over every pair $(\mathcal{A}, \mathcal{B})$ of arguments in V. Whenever the conclusion of $\mathcal{A}$ contradicts the conclusion of $\mathcal{B}$ or some subargument $\mathcal{C}$ of $\mathcal{B}$ and the weight of $\mathcal{A}$ is greater or equal than the weight of $\mathcal{B}$, then an attack has been found and $\mathcal{C}$ is the point of attack.

Computing the grounded extension: Once the argumentation framework is computed as a directed graph G, the algorithm explained in Sect. 3.1 is run. In practice, this algorithm is a variation of a topological sort. This requires finding all the roots of the graph (i.e. vertices with indegree equal to 0), printing these vertices, deleting them and its successors, and then repeating the process until all of the vertices have been processed.

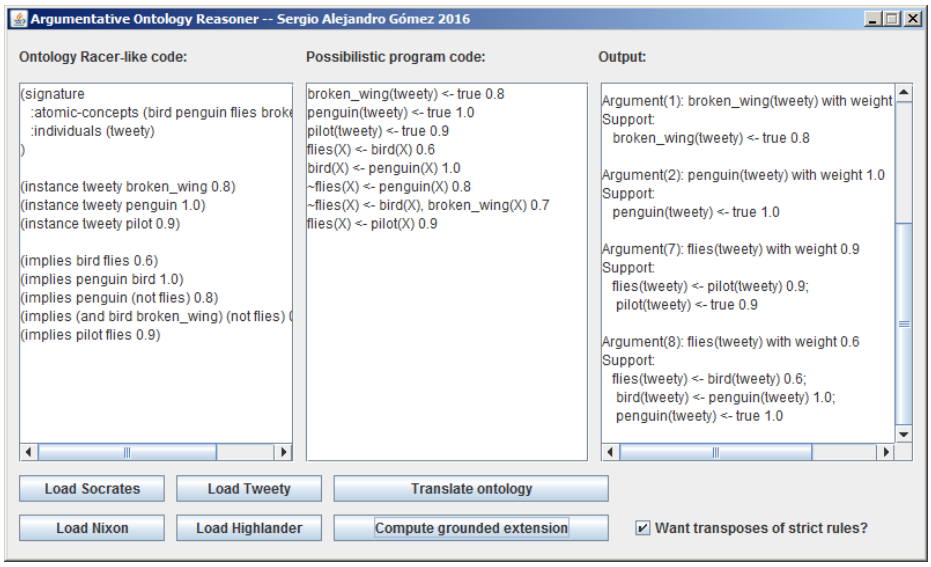

Fig. 3. User interface of the engine for reasoning with possibly inconsistent possibilistic ontologies

5.3 Analysis of Time Complexity

We now present an analysis of the complexity of the algorithms discussed. Let us suppose that the ontology has n inclusion axioms and m assertional statements. Each one of these will generate exactly one rule during the translation into possibilistic logic programming unless contraposition of rules is preferred. In such a case, each strict rule with a distinct atoms will exactly generate a transposed rules, thus increasing the size of the resulting knowledge base.

For simplicity, we will restrict the discussion to logic programming rules with exactly one variable that result from the interpretation of weighted ontologies (even when the system presented is capable of handling reasoning with rules having atoms with several variables). So if there are n rules, k names of constants and if each rule has exactly one variable, then for each rule the propositionalization generates k ground rules and thus the size of the propositionalized knowledge base is $\mathcal{O}(nk)$. Instead if each rule scheme has at most v variables, then the propositionalization process generates k^v ground rules for each rule scheme. If the original kwowledge base has n rules, then the propositionalization generates $\mathcal{O}(nk^v)$ ground rules.

Once the knowledge base is propositionalized, the argument base has to be computed by finding the fix-point of the derivation function. Suppose that A is the total number of arguments computed. For computing the attack relation, we need to find all the $(\mathcal{A}_i, \mathcal{A}_j)$ with $i, j = 1, \ldots, A$ such that $\mathcal{A}_i$ attacks $\mathcal{A}_j$. The argument $\mathcal{A}_i$ attacks the argument $\mathcal{A}_j$ if there is a subargument $\mathcal{B}$ of $\mathcal{A}_j$ such that the conclusions of $\mathcal{A}_i$ and $\mathcal{B}$ are complementary and the weight of $\mathcal{A}_i$ is greater than or equal to the weight of $\mathcal{B}$. If each argument is considered as a tree, this process requires to look at each node of the argument. If the tree of an argument has a degree of ramification s (i.e. the maximum number of children/subarguments that a certain node/argument can have) and the height of the tree coincides with the maximum number h of rules forward-chained, then the number of nodes of an argument can be bounded by $\mathcal{O}(sh)$. Besides, determining if an argument $\mathcal{A}_i$ attacks another argument $\mathcal{A}_j$ when h is the height of $\mathcal{A}_j$ and s is the ramification degree of $\mathcal{A}_j$ requires checking $\mathcal{O}(sh)$ literals L to see if the conclusion of $\mathcal{A}_j$ is complementary to L and the weight of $\mathcal{A}_i$ is greater than or equal to the weight of the subargument supporting L.

Building the argumentation framework requires computing a directed graph whose vertices are comprised by the set of arguments in the propositionalized knowledge base and the directed edges are given by the attack relation between arguments. If there are a arguments, each argument has at most s subarguments and h is the maximum number of chained rules, then computing the attack relationship takes $\mathcal{O}(a^2sh)$.

Computing the grounded extension from the directed graph induced by an argumentation framework requires executing the algorithm presented in Sect. 3.1. This algorithm visits each node of the graph once and also each edge of the graph once, then if the graph has n nodes and e edges, then executing the algorithm takes $\mathcal{O}(n + e)$ (i.e. its time complexity is linear in the size of the graph).

6 Related Work

In [8, 10], Gómez et al. reviewed related work concerning the topic of reasoning with inconsistent ontologies. Because of this, we concentrate our efforts on reviewing implementations of argumentation systems based on Dung's semantics.

During the last years, some approaches based on interpreting ontologies as logic programs have arisen [11]. Gómez et al. [8, 9] have exploited this to reason

on possibly inconsistent ontologies into Defeasible Logic Programming (DeLP). This work differs from those works in that we now propose reasoning with arguments comparable by their relative weight via a grounded semantics approach instead of relying on the DeLP semantics.

Bryant et al. [5] discuss their current work on a prototype light-weight Java-based argumentation engine that can be used to implement a non-monotonic reasoning component in Internet or agent-based applications. Likewise, our system is a Java-based implementation of an argumentation engine based on the representation language of the P-DeLP but interpreting P-DeLP with a Dung-style semantics. However, our engine is not based on the Prolog engine even when the knowledge representation language is somewhat similar.

Snaith and Reed [12] present TOAST, a system that implements the ASPIC+ framework. TOAST accepts a knowledge base and rule set with associated preference and contrariness information, and returns both textual and visual commentaries on the acceptability of arguments in the derived abstract framework. The system can be used as both a web front-end and a web service (See `http://toast.arg-tech.org/`). At the present time, our implementation only provides a graphical user interface that can be run a stand-alone JAR file requiring only to have installed a suitable Java run-time environment. TOAST can compute grounded, preferred and complete extensions but our system only computes grounded extensions.

Tamani and Croitoru [13] introduce a quantitative preference based argumentation system relying on ASPIC argumentation framework and fuzzy set theory. The knowledge base is fuzzified to allow the experts to express their expertise (premises and rules) attached with grades of importance in the unit interval. Arguments are attached with a score aggregating the importance expressed on their premises and rules. Extensions are then computed and the strength of each of which can also be obtained based on its strong arguments. The strengths are used to rank fuzzy extensions from the strongest to the weakest one, upon which decisions can be made. Likewise, our approach allows to express the importance of rules using numbers in the unit interval following the path marked by P-DeLP.

7 Conclusions and Future Work

We have presented a framework for reasoning with possibly inconsistent possibilistic description logic ontologies in the description logic programming fragment by interpreting them in possibilistic defeasible logic programming with a Dung's grounded semantics approach to reasoning, which we enacted with a downloadable Java-based implementation. Our current implementation system computes only grounded extensions, computing other kind of extensions is part of our future work. Our implementation is currently able to process a very limited subset of the description logic programming fragment, adding other constructors to the representation language is part of our current research efforts. Besides, the possibility of reasoning with timed representations and integrating our proposal with Semantic Web reasoners are also interesting directions for future research.

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

References

1. Teresa Alsinet, Carlos Iván Chesñevar, and Lluis Godo. A level-based approach to computing warranted arguments in possibilistic defeasible logic programming. In Philippe Besnard, Sylvie Doutre, and Anthony Hunter, editors, *COMMA*, volume 172 of *Frontiers in Artificial Intelligence and Applications*, pages 1–12. IOS Press, 2008.
2. Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook – Theory, Implementation and Applications*. Cambridge University Press, 2003.
3. Trevor J. M. Bench-Capon and Paul E. Dunne. Argumentation in artificial intelligence. *Artificial Intelligence*, 171(10-15):619–641, 2007.
4. Salem Benferhat, Zied Bouraoui, Sylvain Lagrue, and Julien Rossit. Merging Incommensurable Possibilistic DL-Lite Assertional Bases. In Odile Papini, Salem Benferhat, Laurent Garcia, and Marie-Laure Mugnier, editors, *Proceedings of the IJCAI Workshop 13 Ontologies and Logic Programming for Query Answering*, pages 90–95, 2015.
5. Daniel Bryant, Paul John Krause, and Gerard Vreeswijk. Argue tuProlog: A Lightweight Argumentation Engine for Agent Applications. In *Computational Models of Argument: Proceedings of COMMA 2006*. 2006.
6. P. M. Dung. On the aceptability of arguments and its fundamental role in nonmonotonic reasoning and logic programming. In *Proceedings of the 13th International Joint Conference in Artificial Intelligence (IJCAI)*, pages 852–857. 1993.
7. Sergio A. Gómez, Carlos I Chesñevar, and Guillermo R. Simari. Using Possibilistic Defeasible Logic Programming for Reasoning with Inconsistent Ontologies. In Armando Di Giusti and Javier Diaz, editors, *Computer Science & Technology Series. XVII Argentine Congress of Computer Science Selected Papers*, pages 19–29, 2012.
8. Sergio Alejandro Gómez, Carlos Iván Chesñevar, and Guillermo Ricardo Simari. Reasoning with Inconsistent Ontologies Through Argumentation. *Applied Artificial Intelligence*, 1(24):102–148, 2010.
9. Sergio Alejandro Gómez, Carlos Iván Chesñevar, and Guillermo Ricardo Simari. ONTOarg: A Decision Support Framework for Ontology Integration based on Argumentation. *Expert Systems with Applications*, 40:1858–1870, 2013.
10. Sergio Alejandro Gómez and Guillermo Ricardo Simari. Merging of ontologies using belief revision and defeasible logic programming. *Inteligencia Artificial*, 16(52):16–28, 2013.
11. Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description Logic Programs: Combining Logic Programs with Description Logics. *WWW2003, May 20-24, Budapest, Hungary*, 2003.
12. Mark Snaith and Chris Reed. TOAST: online ASPIC+ implementation. In *Proceedings of the 4th International Conference on Computational Models of Argument (COMMA 2012)*. IOS Press, 2012.
13. Nouredine Tamani and Madalina Croitoru. Fuzzy argumentation system for decision support. In *Information Processing and Management of Uncertainty in Knowledge-Based Systems*, volume 442, pages 77–86. Communications in Computer and Information Science, 2014.
14. Matthias Thimm. Strategic Argumentation in Multi-Agent Systems. *Künstliche Intelligenz*, 28(3):159–168, August 2014.